AD-A047 626   ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB   F/G 12/2

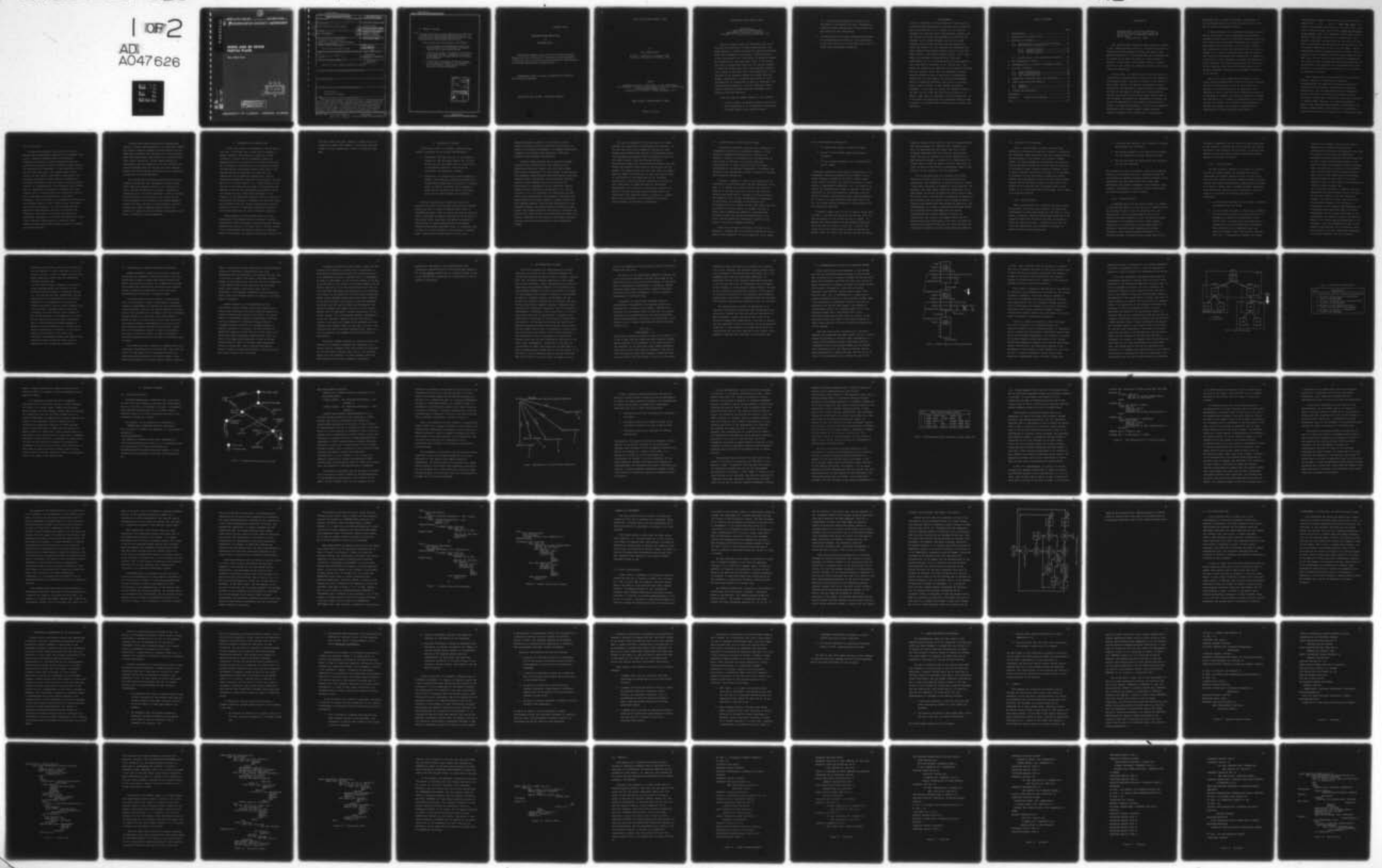
USING AND RE-USING PARTIAL PLANS.(U)

JUN 77   P R DAVIS   DAAB07-72-C-0259

UNCLASSIFIED   R-772   NL

1 OF 2

AD A047626



12

REPORT R-772 JUNE, 1977 UILU-ENG 77-2219

COORDINATED SCIENCE LABORATORY

USING AND RE-USING PARTIAL PLANS

PAUL ROGER DAVIS

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) USING AND RE-USING PARTIAL PLANS,		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		(14) 6. PERFORMING ORG. REPORT NUMBER R-772; UILU-ENG-77-2219
7. AUTHOR(s) (10) Paul Rodger/Davis		(15) 8. CONTRACT OR GRANT NUMBER(s) DAAB07-72-C-0259, AF F33615-73-C-1238
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		12. REPORT DATE (11) June 1977
		13. NUMBER OF PAGES 104
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (9) Doctoral thesis,		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

(12) 113p.

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Partial Plans

Inaccurate World Models

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Partial or default plans are responsible for much of the speed and flexibility of human problem solving. This paper describes a computer planning and execution monitoring system which attempts to use partial or default plans to achieve goals posed to it. The planner is sometimes thwarted and hampered by an inaccurate world model. If the default plan fails to achieve the given goal, the planner attempts to modify the default plan with a local analysis leading to local changes to the failed plan. The modified plan is then delivered to the execution monitor for another execution attempt. Occasionally,

next LB pag

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

097 700

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

a strategy based solely on local considerations is insufficient to result in successful plan modification and execution. In those situations, global analysis frequently reveals that some information can be ignored during plan execution.

There are three key points examined in this research:

1. A local strategy for modifying partial plans can be used usccessfully if it is possible to distinguish a partial failure from a partial success.

2. A fixed local strategy for all goals is not sufficient to distinguish the above. Consequently, a specific searchstragegy must be constructed for each individual goal encountered.

3. A problem solver can largely overcome the detriment of an inaccurate world model if it can experiment in the real world and if it has some idea of how to ignore some troublesome information.

UILU-ENG 77-2219

USING AND RE-USING PARTIAL PLANS

by

Paul Rodger Davis

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259 and in part by the Air Force Avionics Laboratory, Wright-Patterson Air Force Base under Contract AF F33615-73-C-1238.

USING AND RE-USING PARTIAL PLANS

BY

PAUL RODGER DAVIS

B.S.E.E., University of Alabama, 1963
M.S.E.E., University of Illinois, 1970

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1977

Thesis Adviser: Professor Robert T. Chien

Urbana, Illinois

USING AND RE-USING PARTIAL PLANS

Paul R. Davis
Coordinated Science Laboratory and
Department of Electrical Engineering
University of Illinois at Urbana-Champaign, 1977

Partial or default plans are responsible for much of the speed and flexibility of human problem solving. This paper describes a computer planning and execution monitoring system which attempts to use partial or default plans to achieve goals posed to it. The planner is sometimes thwarted and hampered by an inaccurate world model. If the default plan fails to achieve the given goal, the planner attempts to modify the default plan with a local analysis leading to local changes to the failed plan. The modified plan is then delivered to the execution monitor for another execution attempt. Occasionally, a strategy based solely on local considerations is insufficient to result in successful plan modification and execution. In those situations, global analysis frequently reveals that some information can be ignored during plan execution.

There are three key points examined in this research:

1. A local strategy for modifying partial plans can be used successfully if it is possible to distinguish a partial failure from a partial success.

2. A fixed local strategy for all goals is not sufficient to distinguish the above. Consequently, a specific search strategy must be constructed for each individual goal encountered.

3. A problem solver can largely overcome the detriment of an inaccurate world model if it can experiment in the real world and if it has some idea of how to ignore some troublesome information.

Aknowledgment

I wish to thank the Air Force Institute of Technology for providing me the opportunity for graduate study. Thanks are also due my Advisor, Dr. R.T. Chien, for his unfailing optimism, for pratical advice, philosophical insights, and the research facilities and atmosphere to accomplish this research. To Dr. Steven Weissman, who not only asked valuable questions and provided revealing answers as a friend and teacher, but also designed and implemented the basic computer aided decision making system on which this thesis builds, I owe almost unending thanks. To Dr. David Waltz, who provided my introduction to Artificial Intelligence and is a continuing source of encouragement, I owe a huge debt of gratitude. To Dr. Sylvian Ray, I owe thanks for serving on my committee and for being so flexible with his schedule. David Chen, who diligently provided support software so that my work could be demonstrated effectively, and Jack Gladin, who provided photographic advice, stimulating discussion, and a place to hide-out, have my special thanks. To Rose Schmidt and Barbara Champagne, I owe thanks for typing, for frequent access to the boss, and for every day good advice. Finally, to my family, Judy, Paul Jr., and Scott, who have endured more of my bad moods and inattention than reasonably could be asked of anyone, I say thanks with my enduring love, respect, and admiration.

TABLE OF CONTENTS

1. INTRODUCTION

Thinking begins first with suggestive but defective plans and images, that are slowly (if ever) refined and replaced by better ones.
[Marvin Minsky, 1974]

Upon encountering a problem or when planning to achieve a goal, humans typically arrive at a candidate solution by a process which seems somewhat careless and inexact. Many detailed conditions or reouirements that can be assumed true or usually can be made true easily are often ignored in making an initial plan. This process produces a default plan for a given situation, embodying many tacit (but often realistic) assumptions.

Partial plans (or default plans) are plans that are constructed or applied without insuring that all support for the operators used in the plan is present in the planner's world model. Despite the apparent carelessness in their construction and application, partial plans are responsible for much of the speed and flexibility of human problem solving. The ability to use partial plans relaxes some of the stringent requirements for throughness in planners and allows the application of very general existing plans to typical, but specific situations. A plan's initial degree of partialness (and ultimate success) depends upon the planner's estimate of the hospitality of the execution environment and its depth of knowledge. Consequently, a quickly produced partial plan may succeed quietly or may fail to achieve the expected results.

In many situations, it is evident that humans cope with partial plan failure, not by immediately trying another candidate partial plan nor by initiating meticulous planning from scratch, but by using the failed plan as a skeleton for modification and re-execution. It is not clear, however, what perspective is used while modifying the failed plan -- whether it is a global view where every possible cause of plan failure is anticipated and investigated, or whether it is initially a local inspection, ultimately reinforced by a global viewpoint. This paper suggests that the latter view is more consistent with the use of default plans, and it exhibits techniques necessary to and examples illustrative of its success.

Many of the plans constructed by humans in solving simple everyday problems have a definite form and structure. The basic building block or unit of this structure is a sequence of actions followed by an observation. The execution of such a plan unit first requires the actions to be performed in sequence, and then the observation is made. The observation is, in effect, a postcondition [14] to the actions. Its purpose is to confirm that the sequence of actions has produced its intended or expected effect. Representative of plans that are constructed by concatenating a number of these basic units are recipes for cooking (mix ..., add ..., stir ..., <u>bake until brown</u>), and starting an automobile (put the key in the slot, push in the clutch, press the brake, turn the key, <u>listen for the sound of the engine</u>). When a plan composed of these basic units is executed, the primary strategy is to execute each unit in sequence. Departure from one basic unit and entry into the next is conditional upon satisfying the postcondition or the terminating observation for each unit. But people exhibit considerable flexibility when using this strategy. They are frequently able to complete the entire plan execution, even when one or more of the unit postconditions is apparently not satisfied. Some inconsistency between postconditions and preconditions (when apparently all requirements have been satisfied, but confirmation by observation is not possible) is frequently tolerable.

This research effort has concentrated on showing that a computer system can incorporate two techniques (experimentation and ignoring the offending observation, hereafter called procedural verification and skipping, respectively) into the hierarchical planning approach to provide flexibility in modifying and executing simple, sequential, linear plans. These plans, which may be partial or default plans, often can be executed successfully despite some postcondition-precondition inconsistency. The example problem domain is the detection and correction of simple failures and error conditions in a somewhat idealized twin jet aircraft.

Procedural verification and skipping are ways of dealing with postconditions (sensors in the airplane) which do not respond as expected during plan execution. Procedural verification involves interrupting the plan analysis and diagnosis process to perform a simple dynamic but non-destructive experiment. It takes advantage of the the fact that portions of the problem domain are not static, but obey certain cause-effect relationships. But procedural verification, when faced with incomplete or inaccurate information, does not always provide conclusive results. It is essentially a local technique, which usually does not incorporate any measure of progress toward the original goal. To provide additional flexibility, there is also a requirement for complementary, more global techniques, short of giving up, to successfully contend with plan execution failure. These include trying a collateral (but locally non-optimal, i.e., use a longer plan that embodies another approach) approach to satisfying the observation requirement, or ignoring the observation requirement altogether, if all other alternatives have been tryed. Skipping a postcondition is limited to those situations where identifiable progress toward the original goal can be achieved.

To assure that these techniques are appropriately applied, a canned search startegy is not sufficient. Rather, the possible relations between the plan failpoint and the underlying situations (the preconditions of the failpoint goal) that might cause a plan failure are classified into types. When a plan fails, a local search strategy is constructed using the semantics of the plan failpoint and general knowledge about the failure type. This strategy applies the relevant search alternatives in an order which is perceived to be most efficient.

This system usually proposes another partial plan to replace a failed plan. The replacement plan may contain either additional detailed actions or an instruction to ignore the past failpoint while trying to execute the replacement plan. The latter represents a shift in strategy, focusing attention on the original goal instead of the current failpoint. This contrasts with program debugging systems [15] which are able to shift attention to global problems only after a complete success locally. The technique is especially useful whenever the planner's world model is suspected of being inaccurate.

2. DESCRIPTION OF A PARTIAL PLAN

To solve the problem of transforming a failed partial plan into a sufficient plan, a local search strategy is chosen initially. The reasons for preferring a local strategy to a global one are illuminated greatly by examining the form or structure of a typical partial plan. For our purposes, a partial plan is a sequence of actions and observations. The actions present (there may be some missing) may not be sufficient to insure that the observations can be satisfied. Additionally, certain supporting observations (subgoals) may be missing. The plan may not be in uniform detail, i.e., some portions of the plan may be more detailed than others. We assume that the sequence of actions and observations are properly ordered with respect to time. Though this assumption is not necessary to the techniques presented (indeed, a method for dealing with the problem of unordered plan steps can easily be incorporated into the overall strategy either locally, globally, or both), it does focus attention on causes of plan failure that have not been so thoroughly explored.

The philosophy of using partial plans is to make maximum use of the normal state of the world to obviate unnecessarily detailed planning. Partial plans will succeed whenever the world is in its usual state. A partial planner is more appropriately described as shallow or simplistic rather than ignorant, as intuitive rather than careless, and as frugal rather than lazy. Roughly, a partial plan is a subjective, common sense summary of the usually important things to do and observations to make to accomplish some goal.

3. QUESTIONS OF STRATEGY

Using partial plans in a computer problem solving system is motivated by two diverse considerations:

1. Efficiency. Over the long run, it is prudent to account for the way things usually are. No plan should have any more detail than is required for its successful execution. Plans should be sufficient, not necessarily complete.

2. Flexibility. No problem solver which operates in a dynamic, even moderately complex environment can realistically foresee every alternative. If generic plans for typical situations can be adapted to solve specific problems that occur, the problem solver has an added dimension of performance.

The first consideration implies that the search algorithm used when a partial plan fails should be somewhat predictive; that it should be hesitant about depth of knowledge and detail, that it should look ahead at least one level to determine a plausible strategy for the problem at the current level. Conversely, the second consideration implies that the search algorithm must be ultimately concerned with adding sufficient detail. It recognizes that, if there is to be any success in using partial or default plans, instantiated problems must be linked to the appropriate generic exemplar in some manner, and the exemplar eventually must be sufficiently specified to be useful in the current environment. Any computer system which attempts to use default plans must have a method for balancing these opposing considerations.

Existing systems have not had to face this problem. Currently, most problem solvers and planners use the hierarchical approach. However, there are some evident difficulties in applying it to this problem. In the same way that automatic backtracking has been described as blind when reasoning about alternatives, pure hierarchical planning might be labeled somewhat naive and blind with its predilection for expanding a goal to additional details. Whether the goal hierarchy is established by an a priori ordering [13] or by a procedure [14] associated with each node of the representation, the approach to satisfying the goals essentially has been singular -- expand the goal to more detailed subgoals. This has been satisfactory in many situations because the problem domains typically have been simple. The quandary about how much detail is sufficient has never arisen. Since details have always been the answer, the singular commitment of the planner to more detail generally has been locked into the structure of the hierarchical planner.

The works of Weissman [18] and Sacerdoti [14] (where planning and execution are intermingled) are indicative of the fact that this fixed strategy for satisfying a goal becomes less effective when applied to multifaceted domains. Both Weissman and Sacerdoti adopt a compromise solution to the detail quandary. They each provide the basic hierarchical planner with more fixed alternatives. But each retains the strategy firmly fixed in the structure of the planner. This approach does not carry over easily to domains where the number of alternatives at each level may be large or the structure of the domain may not be precisely hierarchical. In such situations, there may be required as much intelligence to choose and test the appropriate alternatives as is required to achieve the original goal. Hence, to provide the proper balance between detail and abstraction, some adaptive solution is needed -- a planner which uses the semantics of the problem to construct a unique strategy for each goal it encounters.

3.1. Justification of Local Search Strategy

Modifying and extending a partial plan, especially in incompletely specified or inconsistent domains, is more related to plausible reasoning than to deduction. In deduction, failure or inconsistency is the death knell of the process. By contrast, in plausible reasoning, inconsistencies or things that do not work are merely difficulties to be overcome. Consequently, the process that attends to plan failure must be have a reflective, flexible, and interactive personality.

Typically, a default or partial plan results from an abstraction of some process that has been successful in the past. It is usually not a generalization, but rather is abstract in the sense that few if any of the underlying decisions and bases used to construct the plan originally are preserved in its current representation. Consequently, in this thesis, no knowledge of the decisions that influenced the form or content of the plan (other than what is evident in the plan itself) is assumed. The plan is given, with no supporting comments or clarifying data structures except the currently perceived state of the world. The only additional information available is the result of the plan's attempted execution.

Given only such sparse information, the task is to determine a strategy that will succeed in making the failed partial plan executable. At first examination, there appear to be three possible alternatives:

1. To replan from scratch, carefully in detail.

2. To use a local strategy keyed by the plan's failpoint.

3. To use a global strategy to try to determine the plan's intent.

Lacking any knowledge of the planner's motivations or its incremental planning decisions [6], any attemt to use a global strategy to determine the plan's "intent" seems either trivial or futile. The intent of the plan is either evident or impossible to determine, i.e., its intent is to satisfy the postconditions present in the plan. The first alternative is not in philosophical consonance with the original purpose of using partial plans. Some amalgamation of (1) and (3) is less attractive than either alone. See 3.3. for a more detailed discussion of the deficiencies of the hierarchical approach.

Primarily though, both (1) and (3) seem to ignore what is known -- that the plan failed at a particular point and that the planner did not intend for this to happen. It appears that these practical and philosophical objections can be avoided by trying to correct the conditions that caused the current failure and by trying to execute the modified plan. Of course, this approach may lead to another execution failure. Hence, further analysis and modification must be anticipated, but deferred until necessary, i.e., until the modified plan fails again. Not only is this procedure consistent with the concept of using partial plans, but also it involves only a local search prior to each execution attempt -- a search for a limited goal in a limited space to find the cause of failure. Admittedly, at worst (if there is more than one simultaneous cause of plan failure), it may involve a few local searches.

Use of the failed plan as a skeleton for further planning is further justified by the realization that considerably less information is usually required to modify a plan than is necessary to construct a plan initially. The available plan steps supply additional constraints, and to some degree, can be used to define the initial search space. The search space is initialized by attempting to execute the given partial plan. But using a local strategy to modify a skeleton plan imposes some requirements and discipline. The representation of a plan as a potentially multipass process is considerably more complicated than the typical representation as a simple sequence of actions and observations. And finally, as Goldstein points out [5], searching the local plan space is not always sufficient to discover the required corrections. The local approach must ultimately be backed up by some global analysis capability.

3.2. What Has To Be Recognized

Using a local strategy to modify a partial plan skeleton for re-execution requires an ability to recognize and categorize events precisely. Such gross categorizations as plan failure or plan success are virtually useless. Indeed, the local strategy for modification and execution must be completely determined from the observed gradations of difference between succesive execution attempts. Hence, to have a sensitive and flexible strategy, it is necessary to recognize not only both partial success and partial failure, but also degrees of each. The distinction between partial failure and partial success in this case is not a subjective one resulting from a different point of view. Rather, it is a prime determinant of strategy, and it needs to be precisely defined.

3.2.1. Partial Failure

When a postcondition for a sequence of actions cannot be satisfied, plan failure has occurred. The unsatisfied postcondition is called the failpoint of the plan. In this system, a compound, multiple step strategy which has as its purpose to make local changes to the plan, is applied to remove the cause of failure. A partial failure (of this local plan modification and re-execution strategy) is defined by the following conditions:

1. A plan has been modified, and an attempt to execute the modified plan has failed.

2. The current failpoint is the same as the failpoint for the immediately previous execution attempt.

3. The local strategy for dealing with this failpoint has not been exhausted.

This situation is easy to recognize, requiring no semantic analysis of the failed plan and no searching through the world models of numerous previous execution attempts. How to proceed with the plan modification has already been determined; the process is embodied in the remaining local strategy. Continuation is achieved by simply reactivating the remaining elements of local strategy.

3.2.2. Complete Failure

A complete failure (of the local strategy) is defined in much the same manner as a partial one. A complete failure has occurred if both (1) and (2) above are true, and additionally, if the local strategy for the current failpoint has been exhausted. As for the partial failure, recognition of the complete failure is easy. However, deciding how to proceed with the plan modification is difficult, requiring global reasoning and strategy formulaton rather than the simple continuation of an existing strategy. Alternatives may include searching for alternate, independent ways to achieve the goal represented by the failpoint, analyzing the world model and the plan to determine if the failpoint can be ignored, or applying some analysis to find and remove unwanted interactions between operators and primitive actions.

3.2.3. Partial Success

The most difficult situation to recognize is a partial success, not simply because the diagnoser may have to investigate the complete execution history of the failed plan, but also because there are numerous degrees of partial success, and because these partial successes may be achieved as a result of either local or global strategy. Recognizing these differences requires careful quantization. Partial success is defined by the following collection of conditions:

1. A plan has been modified, and an attempt to execute the modified plan has failed.

2. (a) The current failpoint is totally new, different from any failpoint that has occurred previously in attempting to execute the predecessors of the current plan. To continue the plan modification, a completely new local strategy must be developed. What information to be remembered about past execution attempts (data, restrictions, confirmed data, etc.) is determined by whether the current

failpoint was a member of the original plan or whether it was inserted during previous modifications of the plan. Modification of the plan may result in both actions and additional postconditions being inserted into the failed plan. If one of these inserted postconditions is not satisfied during the subsequent execution attempt, then a failpoint has occurred at a more detailed level than the level of the previous failpoint. Hence, there is the possibility of the previous failpoint occurring again later. If it does, a complete history of the process will be required to fashion a viable strategy. <u>OR</u>,

(b) The current failpoint is different from the immediately previous failpoint, but is the same as one that has occured at some past execution attempt. This condition occurs when one of the preconditions (because of its structure or properties) for the failpoint gets developed and investigated more thoroughly than its brothers. Consequently, in the current world model of the modification and execution process, the degree of detail is not uniform among preconditions of the failpoint. This is indicative that some of the support (preconditions) for the failpoint (which has reoccurred) has been established successfully by the local approach. The local strategy associated with the most recent past occurrence of this old failpoint is still applicable. But to be useful currently, it must be updated with new world models containing results from the intervening execution attempts. <u>OR</u>,

(c) The current execution attempt is a result of global strategy wherein the failpoint for the previous execution attempt is completely ignored, i.e., all actions have been accomplished, but the previous failpoint postcondition is not checked (purposely -- the assumption is that does not respond accurately to the real world stimulus), and a new failpoint, different from any occurring previously, has been reached. Local strategy for the current failpoint is developed as if the plan failure were the initial one. What restrictions apply in future execution attempts depends on whether the current failpoint is a brother of the ignored postcondition or whether it is a related superior (with respect to the domain goal hierarchy). Distinguishing these last possibilities requires analysis of both the syntax and the semantics of the current plan representation.

3.3 Insufficiency of Standard Hierarchical Approach

Because modifying a partial plan involves using the failed plan as a skeleton, direct application of the pure hierarchical concept to these tasks seems promising. But before concluding this finally, it is imperative to review the strength and liabilities of the hierarchical planning concept, especially as they relate to the distinctive characteristics of partial or incomplete plans.

The primary gain from and interest in using partial plans is the ability to focus on sufficiency of planning instead of completeness of planning. The pure hierarchical planning approach, while it can produce skeleton or outline plans to any specified level of detail, does not distinguish between subgoals at a particular level with respact to degree of detail. In other words, at any given level, the hierarchical planner's representation of a plan has every goal exanded to exactly the same level of detail. Hence, while the hierarchical planner is suited to producing a plan that is uniform in detail, it does not facilitate developing separate portions of the plan to different levels of detail.

But differing levels of detail for separate goals is an identifying characteristic of a partial plan. Goals that are likely (in some sense) to be troublesome typically are supported with much detail by the partial planner, while those goals expected to be "easy" receive cursory attention.

Hence, a brute force, global application of the hierarchical approach to modifying a failed partial plan would necessitiate and be equivalent to an expansion of every goal in the plan, at least to the level of the most detailed subgoal in the partial plan. Even this may not be sufficient if the cause of plan failure lies below the the lowest detail level present in the partial plan. This is potentially a very wasteful computation, and performing it would violate the expressed desire of planning in as little detail as possible.

Another difficulty in the application of the hierarchical concept becomes evident upon comparing its typical goal hierarchy to the goal hierarchy necessary to support reasoning about partial plan failure. In a given domain, a hierarchical planner's effectiveness is derived from the existence of a hierarchy of goals based on increasing amounts of detail and increasing degrees of operator constraint. In contrast, the goal hierarchy to support the analysis of plan failure is more likely to be structured around cause-effect reasoning than levels of detail. This means that moving down a level in the goal hierarchy may not result in finding a more constrained, easier to achieve goal. Hence, the pure hierarchical planner, without a proper goal hierarchy, cannot bring its full power to bear on such a problem.

A related but distinct problem reveals itself when the structure and content of a partal plan is considered. In transforming a failed partial plan into a sufficient plan, it is desireable to adopt an approach which is independent, at least to some extent, of the form and content of the plan to be modified. To the present, It has been assumed that the partial plan steps are in the proper sequence and that the only difficulty (excluding the possibility of an inaccurate world model) is the possibility of there being missing plan steps. We have assumed nothing about which steps (relative to the goal hierarchy) are missing. It has been suggested previously that the level of detail and consequently the choice of subgoals that appear explicitly in the initial partial plan are determined, perhaps subjectively, by the partial planner. For a given partial planner, especially if it has any learning or adaptive ability, the form and content of its plans will change as its perspective of the problem domain changes. Hence, at any time, the form of the resulting plan may be decidedly unhierarchical. Hence, as a first alternative, the standard hierarchical approach is unattractive.

Ultimately, whatever approach is adopted must have the capability to produce a complete plan, beginning with a partial plan as a skeleton. Most of the previous criticisms of the hierarchical approach apply only to its exclusive global use. Its inclusion in a local strategy, where the above pitfalls do not exist, remains an attractive possibility. Additionally, the incorporation of the hierarchical approach as one of the alternatives (hopefully not as _the initial_ alternative) in a global strategy is also a valid possibility, especially in a domain that is rich in operator interactions.

4. WHY INTERACTION IS USEFUL

This thesis explores two complications not usually considered in problem solving or planning systems. The initial plan proposed to achieve a goal is usually a partial plan, and the partial planner operates on a world model that can be inaccurate. Either complication alone offers no particularly new problems; Sussman [15] and Goldstein [5] have examined debugging an almost right program, Waldinger [17] has examined the problem of modifying a valid program to achieve a different purpose, and McDermott [9] has examined the problem of maintaining a consistent data base. The two complications together, however, frequently cause confusion and inconsistent situations which are difficult to disambiguate. Consequently, when a plan fails, its failure may be real (the plan is insufficient) or apparent (the world model is inaccurate so that the plan's success cannot be observed). An inaccurate world model may result from imperfect sensing and data transfer between the real world and the system's world model, or because the problem solver may not have exclusive control of the needed resources. In the blocks world, this is equivalent to a second robot moving a block that has been purposefully positioned by the first robot. Consequently, in addition to providing too little detail at times, the planner may include steps in a plan to achieve things already true in the real world, or it may fail to include necessary steps to achieve conditions that are not true in the real world. The degree to which this is done depends on how the problem solver's world model matches the real world.

The portion of the system that attempts to discover why a plan failed (the diagnoser) initially has access to the world model at the time of the plan's failure. If that world model is inaccurate, inconsistent, or incomplete, neither local nor global search may suffice, without constraints provided by additional information, to find a successful modification to the failed plan.

Inaccuracy in a world model inevitably leads to inconsistency. Inconsistency is discovered whenever all preconditions to produce a given effect apparently are satisfied, but the effect cannot be observed. The expected observation is a postcondition of the actions associated with the preconditions. An example of postconditions in the blocks world is a height check after two blocks have been stacked, e.g.,

PUT A ON B

CHECK HEIGHT = 12

Such a post condition is useful to confirm (if either A or B is not a cube) that the blocks have been properly oriented before stacking. In this example, if the post condition is not satisfied, it is clear that either needed preparatory steps prior to PUT A ON B must be inserted in the plan or that the measuring sensor that transfers length data from the real world to the world model is faulty. If the needed preparatory steps are added and the height post condition still is not satisfied, the diagnoser cannot possibly solve the inconsistency from the existing world model. But if the diagnoser knows that the world model potentially is inaccurate and has at its disposal appropriate techniques, it can use the real world to disambiguate the world model. In this example, it might measure a known length in the real world to calibrate its measuring procedure. If that failed to reconcile the difficulty, the diagnoser might carefully examine block A and block B to determine that they each have the requisite properties and dimensions to be A or B blocks.

The important point is that the possibility of an inaccurate world model necessitates that the problem solver consciously interact with the real world to keep the world model consistent. Thusly, many goals that could not be achieved (because some postcondition in the plan to achieve the goal apparently cannot be satisfied) can be achieved and recognized. Or more generally, a problem solver which can interact with the real world need not always discard a plan because it achieves only a portion of its original goal.

5. REPRESENTATION OF A PLAN AS A CONTINUING PROCESS

After modification and re-execution, a plan becomes more than an action/observation sequence to achieve some end. It serves also as a device to update the current world model with information from the real world and a device to impose constraints on attempts to modify and execute related future plans. Consequently, a plan that has failed execution, been modified, failed execution again, been modified again, etc. is represented as a linked list of execution attempts (Figure 1). Each execution attempt names a data base which is incremently different from the one associated with the previous attempt. This method of representation results in a skeleton model [17] of each execution attempt and a distributed model of the modification/execution process viewed as a whole. At any particular moment, the model of the process consists of these linked histories of previous execution attempts and current changes.

Each goal under active consideration is provided a separate and independent data base called a world. A list of worlds corresponding to the goals under consideration is linked to each execution attempt. Associated with each independent world is the unspent remnant of the strategy that created the world. The remaining strategy contains those alternatives to achieve the goal that are yet to be tried. If the strategy fails to achieve the goal at the

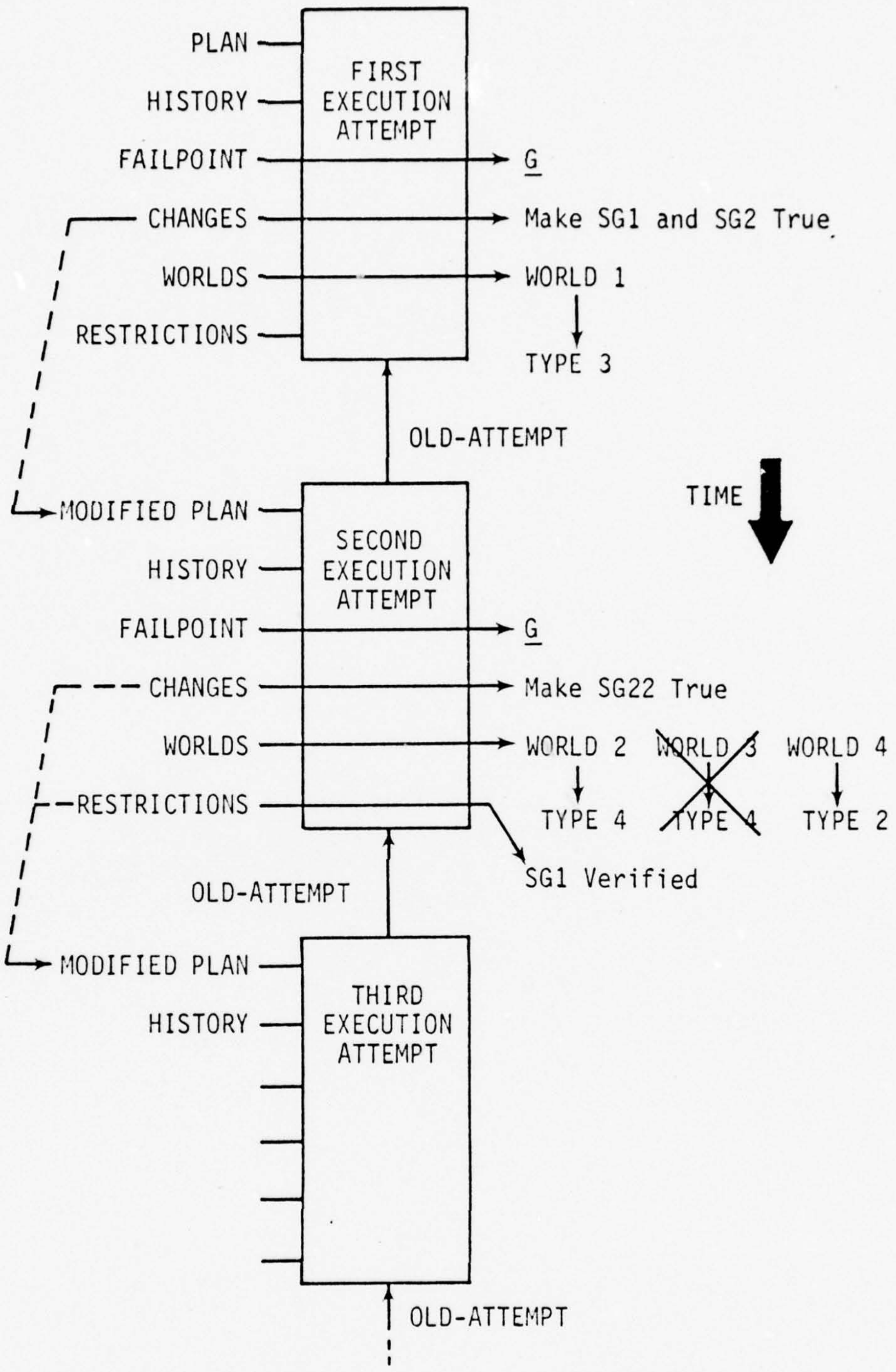


Figure 1. Example Diagnosis and Modification Process

current level of detail, then its purpose is to destroy the world it created, replacing it with a more detailed copy in which the goal can possibly be achieved. The degree of success of the partial plan modification and execution process is characterized by the sequence of the execution attempts and their associated failpoints.

Such a model is especially appropriate for the approach of applying a local strategy first followed by a more global strategy if necessary. Since each execution attempt is explicitly represented, individual attempts can be dealt with independently, and the incremental changes resulting from the application of strategy can be easily identified. Information from previous execution attempts remains available when needed, without the inefficiency of carrying around some massive, layered data structure.

A world is a stack of pointers to other data bases which are a repository for accumulated facts, for recognition demons, and for strategy methods. An independent world is created for each subgoal investigated. Though these worlds are developed independently, the structure is general and flexible enough to allow application of the typical hierarchical approach to identify and eliminate undesireable operator interactions. The hierarchical feature is not implemented in this system. In other systems [14, 15], this function is usually performed by global critics after expansion to appropriate levels of detail. Though such expansion requires a relaxation of our original commitment to having no unnecessary detail, it may be necessary in general to locate and repair all undesireable interactions.

It is this independent development that allows the simultaneous existence of several levels of detail within the model of the modification/execution process (Figure 2). This is necessary and useful, not just to preserve the nonuniform character of the original partial plan that failed, but instead to allow the local strategy (based on the semantics of the current goal) to control the level of detail necessary to achieve the current goal. For example, Figure 2 is illustrative of one situation that makes the existence of several simultaneous levels of detail desireable and necessary. The relationship between the failpoint goal and its preconditions (both SG1 and SG2 are initially unsatisfied) results in WORLD1's being formed and a TYPE3 strategy (Table 1) being constructed and applied. This strategy results in the default plans for satisfying SG1 and SG2 being constructed or retrieved, the original plan being modified to include additions from the default plans, and the execution of the modified plan being attempted. But suppose, for example, that the modified plan fails again at the exact same failpoint as the original plan, even though both SG1 and SG2 now apparently are satisfied. In a quandary (the failpoint goal is not achieved even though its preconditions seem to be satisfied), the system must doubt that SG1 and SG2 actually are satisfied.

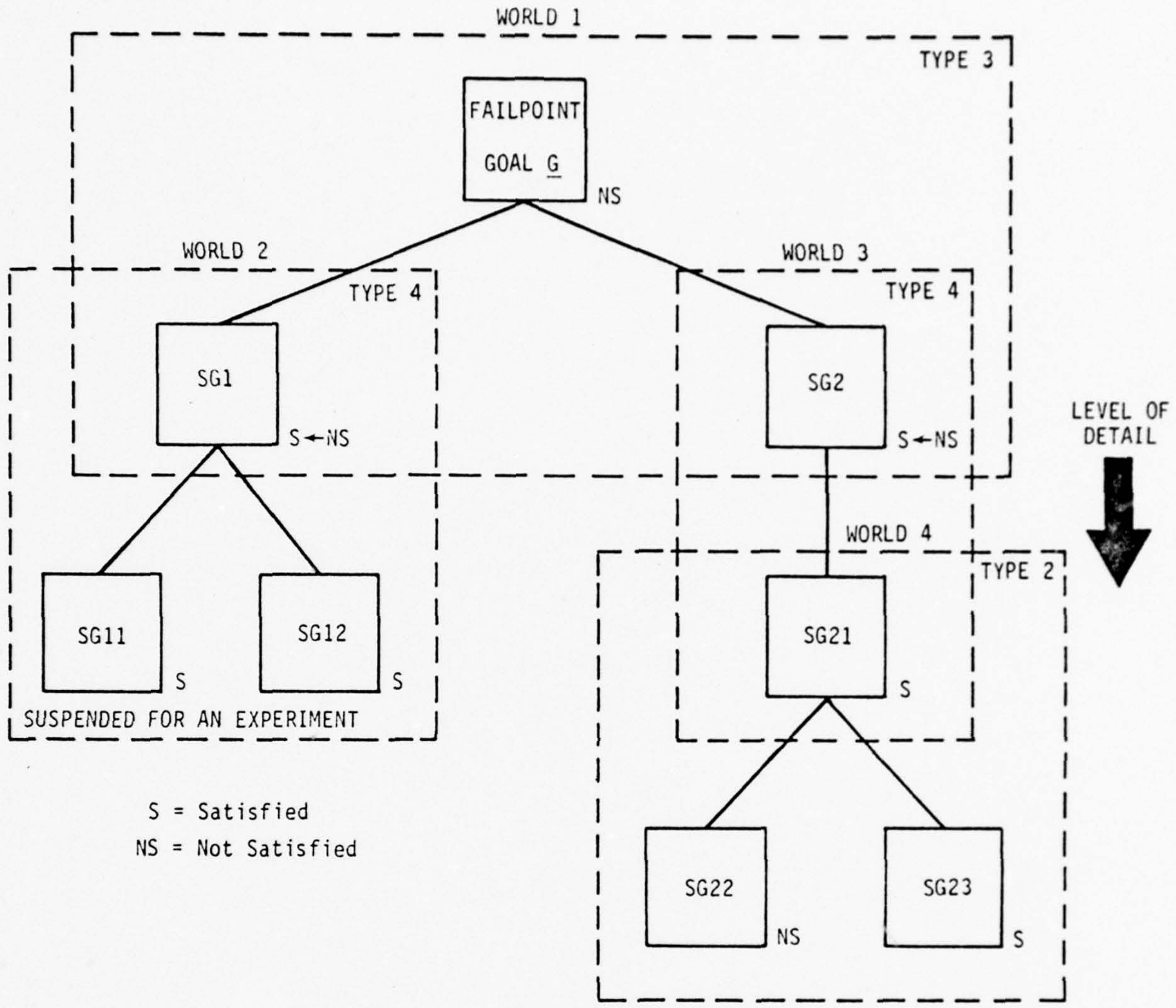


Figure 2. Execution Attempt Model

Table 1. Classification of Sensor Goals

TYPE	CHARACTERISTICS
0	Sensor-type is primitive. There are no preconditions.
1	Goal sensor indicates not OK. All preconditions apparently are satisfied.
2	Goal sensor indicates OK. One or more preconditions apparently are not satisfied.
3	Goal sensor indicates not OK. One or more preconditions apparently are not satisfied.
4	Goal sensor indicates OK. All preconditions apparently are satisfied.

Hence, it begins searching for reasons why they are not satisfied; WORLD1 is expanded, creating independent worlds WORLD2 and WORLD3.

The relationship between SG1 and its immediate preconditions (SG11 and SG22) results in a TYPE4 strategy for WORLD2. In a similar manner, WORLD3 also receives a TYPE4 strategy. For this example, suppose that the structure and properties of SG1 and its TYPE4 strategy result in further investigation of WORLD2 being suspended awaiting the results of an experiment. Further, suppose that the structure and properties of SG2 and its TYPE4 strategy result in the destruction of WORLD3 and the creation of WORLD4. WORLD4 is a TYPE2 world, whose strategy results in application of the default plan to satisfy SG22. If WORLD2's experiment confirms that SG1 is satisfied, the default plan for SG22 and the confirmed information from WORLD2 are integrated into the twice failed plan. The modified plan is then ready for another execution attempt. Note that two distinct levels of detail (WORLD2 and WORLD4) simultaneously exist in the model of the modified plan.

6. BUILDING A STRATEGY

6.1. Local Considerations

Strategy construction is activated when a plan fails for the first time, whenever a plan that has failed before fails at a new point, or whenever a new goal is encountered when expanding the preconditions of an effect. When a failpoint occurs, it is first classified according to the criteria and categories in Table 1.

For example, if the failpoint of a given plan is

```
 (SENSOR-CHECK ((SENSOR28 POSITIVE)) FUEL-FLOW),
```

then in terms of a history (see Appendix I.), the problem is expressed as

```
((FUEL-FLOW POSITIVE)
 (NIL (ENGINE1))(((SENSOR POSITIVE)(NIL (SENSOR28))))).
```

From the Cause-Effect Net (Figure 3), it is determined that the preconditions for (FUEL-FLOW POSITIVE) are (AND(FUEL-QUANTITY AVAILABLE)(FUEL-PATH EXISTS)). If when instantiated and expressed as histories the preconditions are

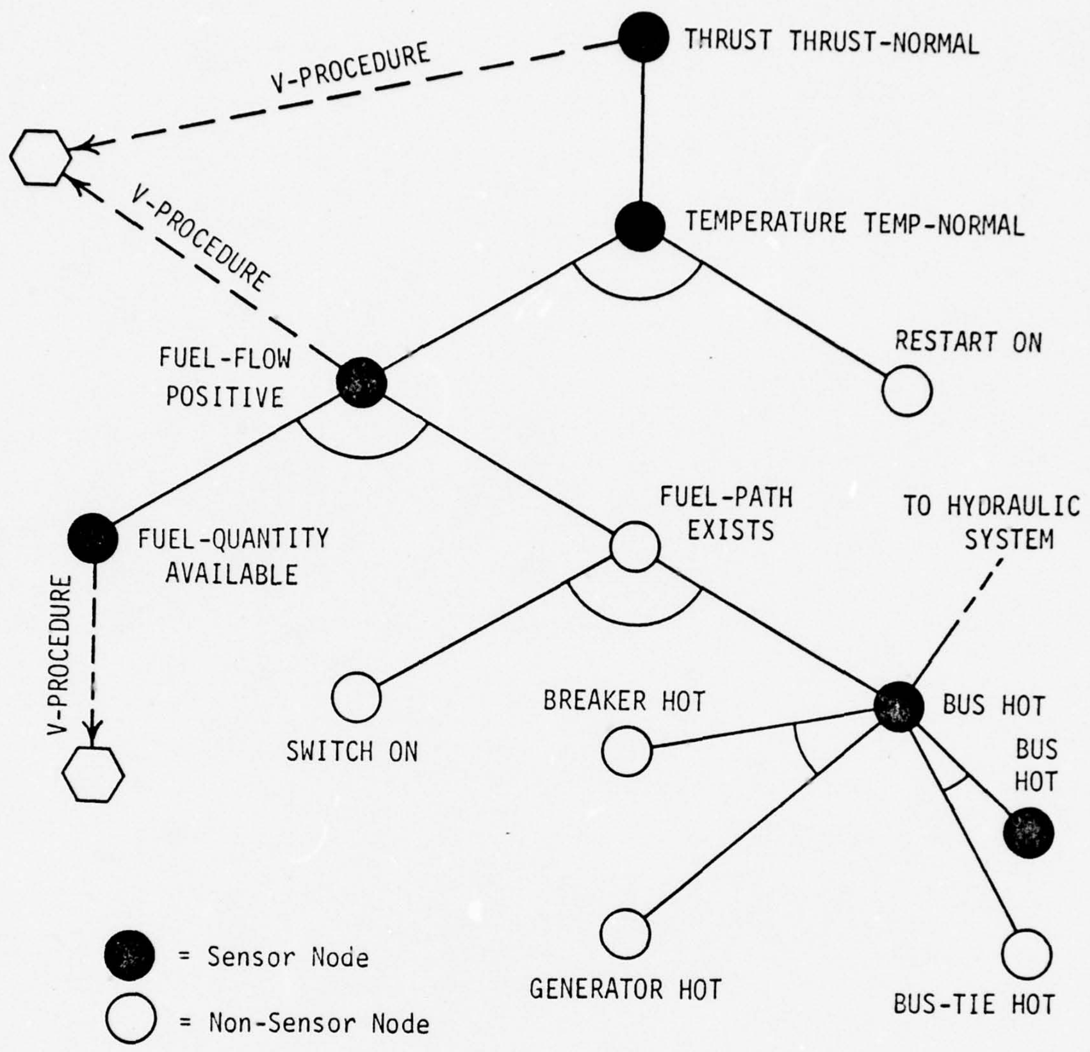


Figure 3. The Cause-Effect Net for the Fuel System

```
(AND((FUEL-QUANTITY AVAILABLE)
     ((TANK3) NIL)(((SENSOR AVAILABLE)((SENSOR24) NIL))))
    ((FUEL-PATH EXISTS)
     (((TANK3 (VALVE5 . ON) (PUMP-FUEL-ELECTRICAL12 . ON)
                                   ENGINE1))
      ((TANK3 (VALVE5 . ON) (PUMP-FUEL-MECHANICAL14 . OFF)
                                   ENGINE1))
     (((SWITCH ON)((SWITCH42 SWITCH38)(SWITCH41)))))),
```

then the problem identified by the failpoint is a TYPE1 problem. Fuel flow is not positive even though the preconditions apparently are satisfied. If one or more of the precondtions could not be satisfied (either no fuel path exists or no fuel quantity is available or both), then the problem would be classified as a TYPE3 problem. Clearly the startegies for dealing with each of these problems differ drastically. A TYPE1 problem causes a quandary (Something obviously is wrong, but I cannot find it!), while a TYPE3 problem provides a clear indication of how to proceed (Apply available knowledge to achieve the unsatisfied preconditions.). In this example, if it is within the planner's capability to satisfy the unsatisfied TYPE3 preconditions, the appropriate steps are added to the failed plan, and execution of the modified plan is attempted.

This example illustrates that the strategy for dealing with a particular failpoint depends on the manisfestations of the failpoint's preconditions in the planner's world model. But the strategy (both the time sequence and the content) also depends on the properties and structure of the precondition that was not initially satisfied. In other words, the strategy depends on the semantic properties of the goal, e.g., whether the goal is a solving a TEMPERATURE problem, or a FUEL-FLOW problem, or a THRUST problem. To successfully and efficiently combine both these influences to produce a strategy, a unifying representation [3,20] and knowledge base is required. A portion of the data structure used in strategy construction is shown in Figure 4.

Once a new failpoint has been assigned a classification (TYPE1, TYPE2, etc.), access is provded to the relevant data structure for constructing a strategy or search algorithm. A general procedure is used to compose a strategy from among the elements of the data structure. It is important to note that only new failpoints are classified and get a constructed local strategy. Old failpoints are handled by global strategy.

The advantages of constructing and tailoring the search algorithm at each for each goal encountered are obvious, especially in rich domains with numerous realistic alternatives. The problem space can be searched in a manner exactly suited to the problem. Most superflous alternatives can be eliminated via a local search and a series of tests. Floundering about in the problem space with an unproductive strategy can be virtually eliminated.

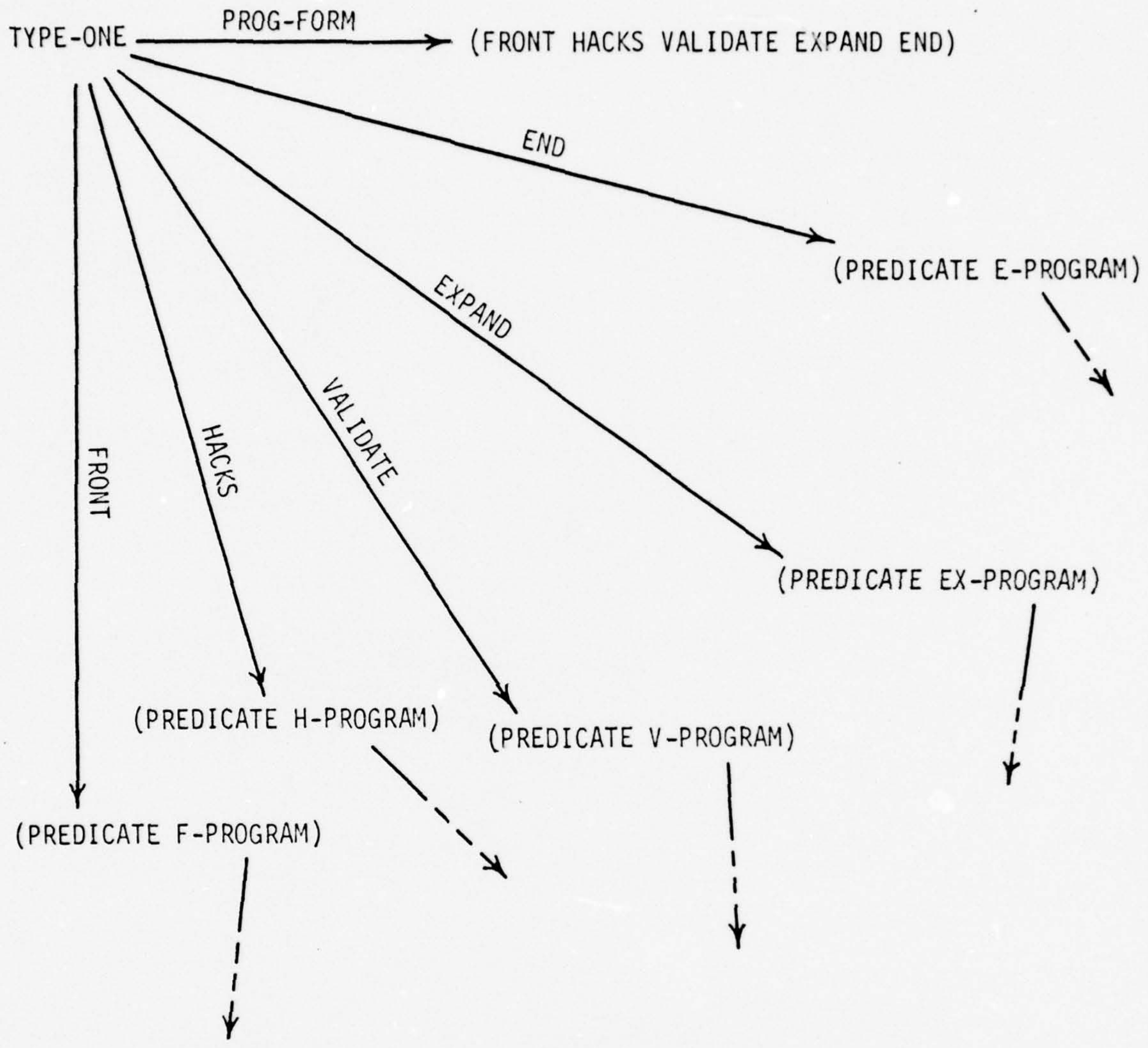


Figure 4. Representation for Search Algorithm Construction

Having a computer system automatically develop and execute a problem solving strategy requires an amalgamation of general and domain specific knowledge. The knowledge base to support the building of semantically based search algorithms must have at least three components:

1. Information about what alternatives are possible in the domain.

2. Information about how to choose a subset of the available alternatives for the problem at hand.

3. Information about how to sequence the selected alternatives.

Additionally, there must be information somewhere (usually implicit in the control structure of the constructed strategy) that indicates how the incremental procedures that make up the strategy are related to each other, i.e., whether the strategy is to be interruptable and re-enterable, or whether it is to be continuous. If the search algorithm includes interruption for execution or experimentation, the interface must support interruption and re-entry. A simple sequential exhaustive strategy, though less flexible and appealing, requires no interface between the modules.

In this implementation, most of the general knowledge is contained in a procedure operating over a small, domain specific data base [19]. Though its contents are domain specific, its structue is general. It contains fundamental modules of strategy (various program segments or modules for accomplishing specific tasks), tests (procedures also) for determining when each of the individual modules might be useful, and procedures for putting the elements in the most useful sequence. For example, there are program segments that can be used to delay further investigation of a goal, segments that result in the performance of an experiment, and segments that expand the goal to create additional subgoals. This representation is illustrative of Rieger's [12] commonsense algorithms and causal selection networks. A difference in this case is that more than one of the fundamental approaches (program segments or modules, in this instance) is usually applicable. Therefore some additional knowledge about combining and interfacing them is usually required.

The procedure that works over this data base has two tasks; one of them is very general, and the other is very specific. First, it classifies the given goal according to the relationship between the goal and its instantiated preconditions. For example, a TYPE1 (Table 1) problem is one where the goal is not satisfied, but where all preconditions apparently have been satisfied. Classification provides access to the list of possibly relevant fundamental strategy elements and module interfaces with a default ordering. A distinct list is associated with each distinct classification type (Figure 5). The sequence of this list is retained unless there is a preferred ordering otherwise specified by the problem semantics. For example, suppose one of the early modules in the original ordering called for the recalibration of the system's measuring device. If that device just recently had been recalibrated, then the module calling for recalibration is probably more useful as a last resort than as an early preparatory step. Hence, its position in the list should be shifted accordingly. Second, the procedure, using the specific knowledge about strategies and the appropriate strategy elements, constructs a stategy for the planner to use to solve the given goal. The resulting strategy is tailored exactly for the problem at hand. In this system, these strategies are implemented as CONNIVER-type IF-NEEDED methods.

Not only is the TYPE of problem important in its construction, but also the characteristics of the specific problem. Individual strategy elements applicable (in general) to the given problem type do not appear in the final strategy unless tests confirm their likely relevance to the specific goal given. For example, if in the above example the domain knowledge base did not indicate that the measuring device had the property of being calibrated, the recalibrate module would not appear in the constructed strategy. The final strategy is the ordered concatenation of

TYPE	MODULE NAMES AND DEFAULT ORDERING					
0	(FRONT	SPECIAL	HACKS	VALIDATE	END)	
1	(FRONT	HACKS	VALIDATE	EXPAND	END)	
2	(FRONT	HACK-FIX	FIX	VALIDATE	EXPAND	END)
3	(FRONT	FIX	HACKS	VALIDATE	EXPAND	END)
4	(FRONT	DELAY	HACK-FIX	VALIDATE	EXPAND	END)

Figure 5. Module Names and Default Module Order for Each Problem Type

the program segments (which survive the relevance tests) and the interface statements between the individual elements of strategy. In this system, the objective of the constructed strategy is achieving the given goal at the current level of detail. If this is not possible, then the system grudgingly expands the goal to greater detail.

This approach to modifying partial plans is an aggregation of some of the ideas from planning, program synthesis, and program modification [8, 17]. The search algorithm that is synthesized is used to modify the partial plan to make it sufficient to solve the current problem. There is little need (though the structure allows it) of backtracking to construct another strategy. The predicates that determine whether a fundamental strategy element is admissible to the final strategy can be quite liberal. Hence, the final strategy usually can be made complete, containing every relevant fundamental strategy element in the order of their perceived usefulness to the problem at hand. Whereas the admission predicates can be liberal, the sequencing procedure can be quite conservative to insure that the most likely alternatives are explored first.

A LISP [11] representation of a portion of the data structure for strategy construction is shown in Figure 6. The PROG-FORM property of the classification is a list of atoms, each of which names a module of code (a subroutine) which may be included in the final strategy. In the absence

```
(DEFPROP ONE (FRONT HACKS VALIDATE EXPAND END) PROG-FORM)

(DEFPROP ONE
         (T (PROG (SENS TAG)
                  (RESTORE TAG CW SENS PROBLEM PRECS)
                  (AND TAG (GO (CAR TAG)))))
         FRONT)

(DEFPROP ONE
         ((GET (CAAR PROBLEM) 'SENSOR)
          ((AND (BRK-CK)
                (SETQ TAG '(TAG1))
                (INSTANCE NIL)
                (AU-REVOIR TAG CW SENS PROBLEM PRECS))))
         HACKS)

(DEFPROP ONE
         ((GET (CAAR PROBLEM) 'V-PROCEDURE)
          ((AND (VERIFY-TRY)
                (SETQ TAG '(TAG2))
                (INSTANCE NIL)
                (AU-REVOIR TAG CW SENS PROBLEM PRECS))))
         VALIDATE)

(DEFPROP ONE (T ((ADIEU))) END)

(DEFPROP ONE (T ((LOOK-DEEPER))) EXPAND)
```

Figure 6   LISP Representation of Strategy Modules

of an ORDER property associated with the classification, the order of the atoms in the PROG-FORM list is a default ordering for the modules that will appear in the final strategy.

In recognition that the final strategy should be influenced both by the goal structure and by its particular manifestation, the module named by each atom in the list is not automatically included in the final strategy. Rather, the entry of the module to the final strategy is restricted by a predicate which is evaluated in the environment of the goal. Only if the predicate is satisfied does the named module become a part of the final strategy. The predicate serves as a coarse measure of applicability of the module to the current problem. It is not failsafe; indeed it is not intended to be. Its purpose is to exclude obviously irrelevant modules. As a final check of applicability (after it has been incorporated into the final strategy), each module performs more careful, detailed tests prior to activating its body. Hence, each local strategy, composed of of the concatenation of all modules whose predicates are satisfied, may be somewhat over specified. Each strategy is at least complete, containing all known and possibly relevant alternatives for dealing with the problem at hand. This completeness gives each local strategy a desireable antecedent character. Once constructed, the strategy does not spend time and effort arbitrating betwen alternatives. Rather, if a proposed element of the final strategy fails or is determined to be inapplicable, the next most likely alternative is immediately provided for application. Consequently, even though the implementation and representation allow it, there is no need for backtracking.

To more fully understand the flexibility of this representation, let us examine some of the modules in detail. The module (Figure 6) FRONT has as its purpose to declare the local variables to be used by the strategy, to initialize their values, and to establish the proper continuation point in the strategy if it has been previously entered and exited for execution of a modified plan.

The purpose of the module HACKS is indicated by its name -- to apply any default or remembered quick fixes so that the perceived problem may be disposed of promptly. If the system has any learning ability, HACKS provides a repository for such information. Actually, the representation provides for hacks to be distributed throughout the final strategy. All hacks need not be quick fixes near the beginning of the strategy. There might be another hack to be used in desperation near or at the end of the strategy. In this implementation, HACKS usually contains procedures which insure that some electrical breakers which might otherwise be forgotten are closed before more drastic causes for a problem are postulated.

The purpose of the VALIDATE module is to locate and to try to dispose of any inconsistency in the planner's world model. It points to a procedural representation which it uses to understand the dynamic behavior of that portion of the problem domain related to the current plan failpoint. Its task is to perform an experiment in the real world whenever a goal's preconditions appear to be satisfied, but the goal does not appear to be achieved. The results of the experiment are used to limit and direct further search for the cause of the anamoly, to delete impossible to observe postconditions from the failed plan, or to change the definition of an important error or failure condition. The latter action is taken whenever a piece of information used in the definition of significant events is determined to be false or unreliable. The result is that the faulty piece of information is ignored when trying to recognize the significant events. If the unreliable information is the only element in the definition of a particular significant event, then the event is no longer recognizable. Consequently, it is removed from the planner's list of significant events to be detected and to which a response is required.

The validate module may limit and direct search by eliminating a particular precondition from consideration as a cause of the inability to achieve the given goal. For example, if a dynamic experiment confirms that a particular precondition already truly is satisfied, then search for the cause of the goal failure is directed to continue elsewhere. Not only is the immediate precondition removed as a candidate for further investigation, but also all logical consequences of its truth also are removed. This may result in a significant reduction in the potential search space.

More impressively, the validate module can often distinguish truly significant events from events that are only apparently significant. For example, if a dynamic experiment fails to return the expected effect, but results in one or more predicted dynamic effects which logically imply the expected (but apparently non-occurring) effect, then the system concludes that the expected effect did occur, but was not observable because of imperfect sensing. In the airplane, this means that it is not necessary for the system to look for the cause of an apparent error condition when there is no error condition but a sensing error (resulting in an inaccurate world model) instead.

The EXPAND module is activated only when the local strategy has determined that a goal cannot be satisfied at the current level of detail. It uses the Cause-Effect Net (Figure 3) directly to direct the inspection of the preconditions of the unachieved goal in the same way that the goal itself was initially examined. The preconditions of each precondition are instantiated, and the problems at the expanded level of detail are classified according to the criteria in Table 1. It is important to note that, because the world model may be inaccurate, no goal-precondition combination can be ignored when expanding and classifying. Even those goal-precondition combinations that indicate all is well must be expanded and classified. Of course, once expanded and classified, all types of problems are not treated the same. The system recognizes that certain combinations are more likely to be the cause of difficulty than others. This distinction is accounted for in the strategy that is constructed for each goal-precondition combination. For example, no TYPE4 goal receives any attention from the planner until all other goal-precondition combinations have had their strategies exhausted at the current level of detail without eliminating the failpoint.

Once the strategy has given up trying to achieve a goal at a given level of detail and expands the goal into subgoals, a new strategy for each new subgoal must be constructed in the same manner as the original local strategy. The express purpose of each local strategy is to achieve the goal at the current level of detail, i.e., to make the plan executeable by adding only as much detail as necessary. If the worst occurs, and the subgoals of the failpoint of the plan must be further expanded, the strategy elements prior to expansion have eliminated or identified all visible problems at the current level of detail. Additionally, any successful validation procedures will have identified more detailed information that can be believed without doubting or expanding.

The purpose of the END module is to signal that the strategy at the current level of detail has been exhausted (serving the purpose of a CONNIVER ADIEU function). But whenever a primitive level has been reached (a TYPE0 problem), i.e., there are no further preconditions to treat as goals for further expansion, the END module also makes some suggestions about the possible causes of the problem. It can take no further action to achieve the particular goal, having reached the limits of its knowledge.

Some example TYPE1 strategies constructed by the system using these modules and the associated representation are shown in Figure 7 and Figure 8. Figure 7(a) shows the simplest strategy. The concept goal, FUEL-PATH, can only be understood and established by obtaining more detailed information. Consequently, LOOK-DEEPER, a function which expands FUEL-PATH EXISTS into subgoals involving pumps and valves, is the only element of the strategy. A more robust strategy (Figure 7(b)) is possible for the concept goal TEMPERATURE since there is a sensor associated with measuring temperature. Therefore, BRK-CK, a function which locates the circuit breaker for the temperature sensor and produces a plan step to close it if it is open, becomes a part of the strategy for accomplishing the TEMPERATURE TEMP-NORMAL goal. If BRK-CK is not successful, i.e., if the breaker is already closed or the plan including the step to close the breaker fails to achieve the TEMPERATURE TEMP-NORMAL goal, then the goal is expanded to more detailed

```
(GOAL =
      ((FUEL-PATH EXISTS)
       (NIL
        ((TANK3 (PUMP-FUEL-MECHANICAL14 . OFF) (VALVE5 .
OFF) ENGINE1)
         (TANK3 (PUMP-FUEL-ELECTRICAL12 . OFF)
                (VALVE5 . OFF)
                ENGINE1)))))
(SEARCH-STRATEGY =
                 (IF-NEEDED STRA4 (TYPE ONE)
                            (PROG (SENS TAG)
                                  (RESTORE TAG CW SENS
PROBLEM PRECS)
                                  (AND TAG (GO (CAR TAG)))
                                  (LOOK-DEEPER)
                                  (ADIEU))))
```

(a)

```
(GOAL =
      ((TEMPERATURE TEMP-NORMAL)
       (NIL (ENGINE1))
       (((SENSOR TEMP-NORMAL) (NIL (SENSOR26))))))
(SEARCH-STRATEGY =
                 (IF-NEEDED STRA2 (TYPE ONE)
                            (PROG (SENS TAG)
                                  (RESTORE TAG CW SENS
PROBLEM PRECS)
                                  (AND TAG (GO (CAR TAG)))
                                  (AND (BRK-CK)
                                       (SETQ TAG '(TAG1))
                                       (INSTANCE NIL)
                                       (AU-REVOIR TAG
                                                  CW
                                                  SENS
                                                  PROBLEM
                                                  PRECS))
                             TAG1 (LOOK-DEEPER)
                                  (ADIEU))))
```

(b)

Figure 7   Example Constructed Strategies

```
(GOAL =
      ((FUEL-FLOW POSITIVE)
       (NIL (ENGINE2))
       (((SENSOR POSITIVE) (NIL (SENSOR32))))))
(SEARCH-STRATEGY
 =
 (IF-NEEDED STRA3 (TYPE ONE)
            (PROG (SENS TAG)
                  (RESTORE TAG CW SENS PROBLEM PRECS)
                  (AND TAG (GO (CAR TAG)))
                  (AND (BRK-CK)
                       (SETQ TAG '(TAG1))
                       (INSTANCE NIL)
                       (AU-REVOIR TAG
                                  CW
                                  SENS
                                  PROBLEM
                                  PRECS))
             TAG1 (AND (VERIFY-TRY)
                       (SETQ TAG '(TAG2))
                       (INSTANCE NIL)
                       (AU-REVOIR TAG
                                  CW
                                  SENS
                                  PROBLEM
                                  PRECS))
             TAG2 (LOOK-DEEPER)
                  (ADIEU))))
```

Figure 8   Example Constructed Strategy

subgoals by LOOK-DEEPER.

Even more alternatives are possible for FUEL-FLOW (Figure 8) which can be verified by an experiment. Hence, VERIFY-TRY, a function which uses the Cause-Effect Net to select and perform an experiment is included in the local strategy.

The process results in what might be called second order planning, i.e., planning about how to plan. Usually the process of reasoning about and planning a strategy for modifying an existing plan is simpler than reasoning about how to solve the problem from scratch. Indeed, the amount of required knowledge about programming tactics seems small [8], especially when compared to the amount of domain knowledge already in use.

6.2. Global Considerations

A major issue in debugging and hierarchical planning systems has been how to maintain a global view of things, i.e., how to insure that the numerous individual changes made on the basis of local decisions make sense together globally. In debugging systems [5, 15], problems are typically small programs that perform satisfactorily when simulated in isolation, but perform unsatisfactorily or not at all in context. A top level or supervisory perspective is required to adapt the program module (which has been proven in private) to the current context. In hierarchical planning systems, the requirement for a global view stems from the very nature of the hierarchical concept. Its basic strategy is to create a plan as cheaply as possible by ignoring a lot of information. If the constructed plan proves to be insufficient, it is expanded to more detail on the basis of local considerations [14]. In such a system, some effort must be dedicated to overview to insure that redundant actions and incompatible action sequences are avoided. In both debugging and hierarchical planning systems to date, the basic concern of the overview function has been to insure a harmonious relationship among the actions in a plan or program.

My basic approach to the problem of modifying a partial plan is somewhat analagous to top level [5] debugging strategy. I try to make local changes (really as local as possible) to eliminate the possibility of a plan failing again at the same point as previously. A global perspective is necessary in cases where these local modifications are not successful in removing the failpoint, especially when the plan being modified is a partial plan.

Any effective global view, especially in the context of evolutionary plan modification, requires a definitive answer to the question, "Am I making progress toward the original goal?". The problem of sequencing plan steps properly has been extensively explored [13, 14, 15, 16, 17, 18] as a portion of the global view, but the remainder of the pervading problem (how to measure progress toward the given goal) generally has been neglected. Certainly, a consideration of which plan steps might be ignored or deleted from the plan because they appear useless or counterproductive is a portion of the global view. Another necessary task of the global view is to distinguish between local strategies that should be revived and continued and those that should be rejected in future execution and modification attempts. The strategies chosen for continuation determine what accumulated information to retain and what to purge in the current world model.

There are two distinct considerations in the global strategy for dealing with partial plan failure. The basic dichotomy is formed by whether or not the partial plan has failed once or numerous times. If the plan failure is the initial one, then the relevant procedure for constructing a specific strategy for the current failpoint is employed. If the plan has failed previously more than once, there are two other distinguishing questions: is the current failpoint identical to the failpoint of the immediately previous execution attempt, and if not, has this current failpoint occurred during some previous execution attempt? If the former, then any remaining strategy is revived and continued. If the latter, the strategy associated with the latest previous execution attempt having the same failpoint as the current execution attempt is updated with the current situation and continued. See Figure 9 for details.

Despite the fact that the construction of new or the continuation of old strategy produces only local changes, this approach fosters an effective, global overview in three distinct ways. First, if a failpoint from the immediately previous execution attempt is revisited, this is indicative that the local strategy has not succeeded in any way. Since the global modification process believes the current, local strategy to be complete (and hence ultimately effective), the global direction is to continue the existing strategy to completion. Second, if a previous failpoint is revisited on a (not immediately) subsequent execution attempt, failpoints traversed in the meanwhile must have been among the preconditions (or the support for the preconditions) of the current failpoint. This is indicative that the strategy associated with the latest previous occurrence of the current failpoint has partially succeeded. Since this is evidence of progress, the global direction is to provide current world models to the old strategy and to continue any unspent elements of the old strategy. Third, if all elements of methodical strategy are exhausted, i.e., a primitive level has been reached without eliminating the failpoint, this is evidence that the goal (represented by the failpoint) either is extraneous or that the system's world model is inaccurate at some point. The global direction, at this point is to abandon the attempt to make local changes and to try to "make progress" toward the original goal by

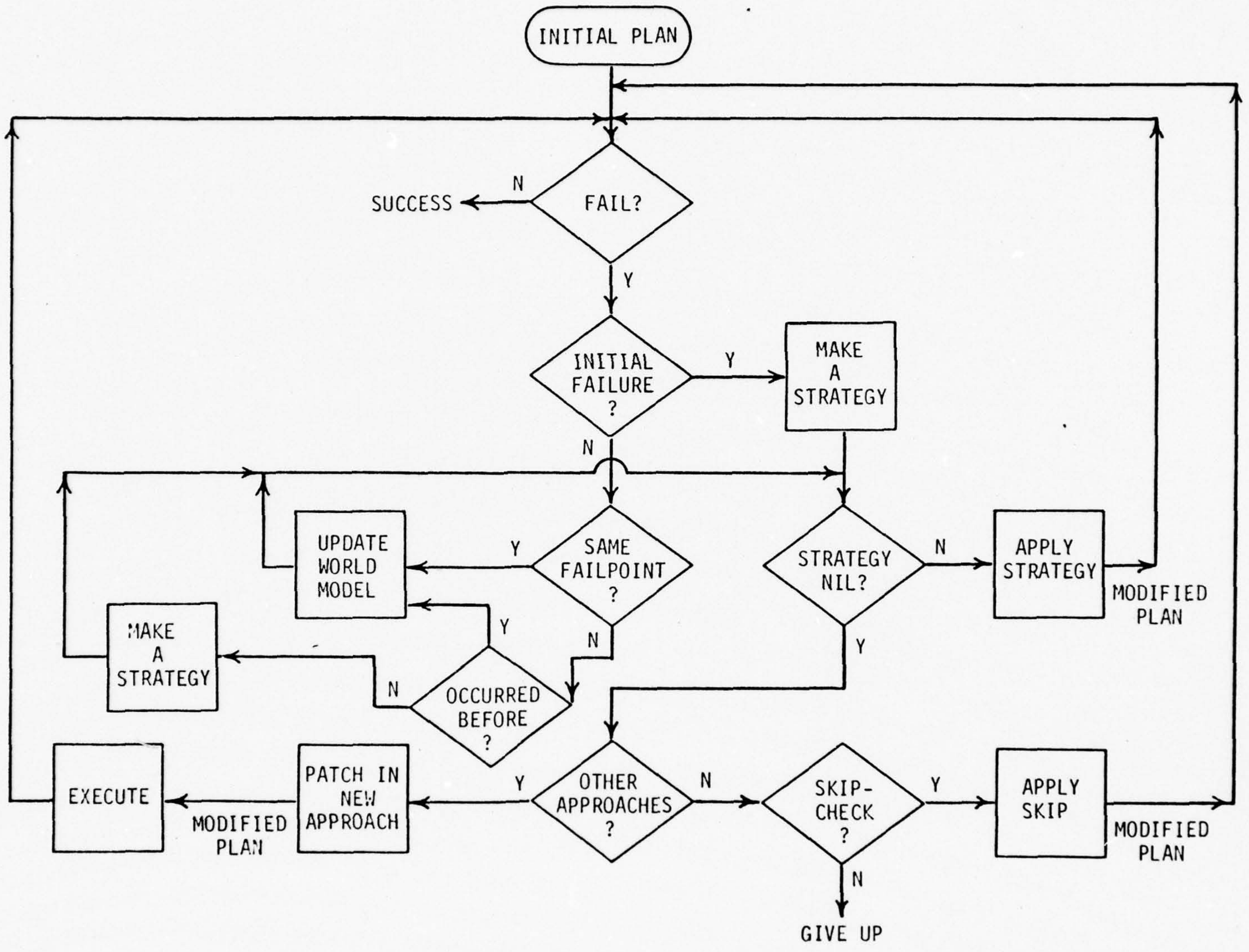


Figure 9. Flow Diagram for Diagnosis

ignoring the failpoint goal. "Making progress" is defined either as attaining the final goal of the original plan or as achieving some father goal of the current failpoint goal.

6.3 The Cause-Effect Net

In the original CADM [1] system, most of the representation of the failure correction process was embodied in procedures. All plans produced by the original CADM system were complete, and the problem domain was completely specified. With these idealizations, the problem was finding a plan; once found, it always succeeded. However, when extending these correction procedures to a lesser specified world model, they sometimes produced a plan that failed to accomplish its goal. If plan failure occurred,it was difficult to determine the cause of failure. Characteristically, the procedural representation was difficult to analyze to determine the capability and span of influence of each procedure.

In order to reason about the plans produced by CADM, an easily analyzable description of the correction process is needed. Some explicit idea of what effect each action produces, of what actions are necessary to achieve a given effect, of what always infallibly is known when a particular dynamic effect is observed, and of the way that the numerous observable effects are related is needed. A portion of the representation developed (called the Cause-Effect Net) for these purposes is shown in Figure 3. The portion shown portrays cause-effect relevant to a simple aircraft engine. It is a net (not a tree) because not every node has a unique predecessor and because some of the attached information

(V-PROCEDURES, in particular) are shared by several nodes.

The Cause-Effect Net serves two distinct but related functions to support reasoning about plan failure. First, it serves as a process description for failure correction. Its structure, in this function, describes the way various effects are related. With the aid of attached procedures, it serves the function of common sense algorithmic knowledge [12] for the airplane domain. Secondly, it serves as a goal hierarchy for the hierarchical planner which expands a node to more detail. When a goal is expanded into subgoals, the net is traversed in a downward direction. Which of the preconditions are chosen as subgoals to be achieved depends on their logical relationship (simple AND/OR in this case) to the goal and on their particular instantiation (whether or not they appear to be already satisfied). Where there is an OR relationship, the combination of subgoals which produces the shortest plan to achieve the immediate local goal is preferred initially. When it is necessary to determine if a particular strategy is making progress toward the original goal, the net is examined in an upward direction.

7. INTERACTION, EXPERIMENTATION, AND VERIFICATION

Problem solving and planning systems have demonstrated an ability to partially understand and manipulate certain microworlds. However, attempts to extend many of the developed principles of modeling facts and the consequences of actions to more interactive worlds have met with some difficulty. In this light, the principle limitations of the initial microworld modeling were the omission of a representation for processes that are not instantaneously achieved once initiated and the omission of all effect causing entities except the robot itself. Consequently, situations which were of interest for their process description could only be represented by a before-after snapshot. Problem solvers which must operate in richer, more interactive worlds [18] have found not only that they must contend with these complications, but also that, if the problem solver's representation of the world's processes is adequate, that cause-effect reasoning is frequently useful in understanding and solving dynamic problems. A far reaching result of this realization has been the endowment of the robot planner with the ability to interact with the real world through experimentation. The results of auxilliary experiments performed by the planner can influence the problem solving process by introducing new information, by purging useless information, and by altering the problem solver's confidence in the accuracy of its world model.

When one is confronted with a problem in the real world, it is frequently worthwhile, before trying to solve the problem, to determine if it is real or only apparent. Likewise, when one is presented with a new item of information to be used in decision making, it is usually prudent to establish the veracity of this item. Analogically, a small pause by a computer system prior to commiting its resources to searching, planning, or deduction is often advantageous.

Previous researchers have recognized this fact in some contexts and have proposed that a computer system in such situations might ask a human user for advice [14], might simulate the fact and its attendant processes in the background [2, 9], or might search its current world model for conflicting and inconsistent data [9]. In varying degrees, each of these approaches has the following disadvantages:

1. The assumption that there is someone available with a better perception of the problem domain than the computer system's world model provides. Such may not be the case in a robot space station, for example.

2. The assumption that the computer system has sufficient accurate information in its current world model to reach a conclusion. This is frequently not realistic [18].

The real disadvantage of these approaches, however, is not that they are unrealistic. Rather, they are too restrictive, tending to focus attention on the system's model of the world and tending to ignore the real world. In many situations, the real world is accessible. Taking advantage of this, Weissman [18] relaxed the distinction between planning and execution. This allowed Hairy Reasoner to assume things and to observe additional confirming information. Yet the real world has further potential for problem solving than as a depository for static facts. Its inherent dynamic character, which has long been considered detrimental to computer problem solvers, and its primal cause-effect relationships can be useful to the problem solver if it is able to interact with, in addition to simply observing, the real world. In fact, as a result of purposeful interaction, the problem solver's observation abilities can then be made more valuable. They become more expectation or goal driven instead of change or difference driven.

In addition to the ways previously discussed, a computer system can interact meaningly with its environment in two ways:

1. By focusing its attention on a dynamic quantity in the real world while expecting it to change in some way.

2. By performing some experiment in the real world and watching for expected results. In the discussion that follows, either of these methods will be called procedural verification.

Expectation is necessary to distinguish uninteresting changes from important changes in a dynamic world. It insures that messages are decoded by the appropriate data bases. It tags and identifies important information as being relevant to a particular problem. It is assumption, waiting to be confirmed or denied. A requisite for generating expectation is a knowledge of the cause-effect relationships that exist in the real world. This common sense information [12] must be captured in the system's knowledge base in a useable form. This frequently requires two distinct types of information -- a model of what things can be done and a characterization of what is known upon completion of each allowable action.

If one is going to the trouble to experiment, he should be prepared to extract the most information for his efforts. An experiment can potentially return three distinct kinds of information:

1. Direct information, obtained from observing the local expected results of the experiment. This information is usually used directly by the system as fuel for deduction.

2. Indirect information, obtained from observing redundant or side effects of the experiment.

3. Secondary information, obtained by deduction from the direct or indirect information. For example, in typical problem solving systems, if an experiment results in the expected effect, then the preconditions for that effect must have been previously satisfied. If this set of facts is not already in the world model, each member of the set may be asserted.

In many situations, this secondary information may be redundant and useless. For example, an operator usually has a set of preconditions that must be satisfied before it can be applied. If it is impossible for the system to satisfy the preconditions of an operator in the plan, plan failure occurs. But in situations where the problem solver's model of the world may be inaccurate, some of the preconditions to guarantee success of an operator apparently may not be satisfied. In the absence of other alternatives, provided that certain very general conditions can be satisfied, the desired operator may be applied as an experiment. If it produces the desired effect when applied, then the derived secondary information may be useful in breaking a log jam in plan execution. Specifically, information obtained in this manner is absolutely credible. It requires no later search or verification to substantiate. Hence, the preconditions of a particular operator are still valuable in uncertain worlds. A priori, they generate subgoals to be achieved, exactly as in a "perfect world" application. A posteriori, they may generate confirmed, certain information.

Procedural verification involves the following:

1. An ability to initiate an action at a particular time in the future. This action usually results in a perturbation of the real world and begins the experiment.

2. An ability to terminate an action at a specified time in the future. This usually returns the world to the quiesent state.

3. An ability to observe and report direct and indirect information. These types of information usually occur as a relaxation of the environment in response to some action.

4. An ability to deduce secondary information from the results of the experiment.

In short, in addition to the the ability to model simultaneous actions and continuous processes [7] regnant in the real world, the experimentor requires an ability to understand and use the results of the experiment.

Procedural verification is implemented by associating a program or sequence of programs with each verifiable concept in the domain. These programs are used by the local strategy that is constructed when a relevant goal is encountered. To be verifiable, a concept must satisfy three conditions. It must have a known cause-effect relationship with some other concept in the domain, it must be dynamic (change with time or perturbation), and there must be a non-destructive action that excites the known cause-effect relationship.

Each program in the sequence has at most the following segments:

1. A segment which uses the relational data base describing the problem domain to bind the program variables.

2. A segment which uses the bound variables to create and activate expectation demons for direct, indirect, and secondary information. This is accomplished by using the variable bindings to alter a general purpose function which writes appropriate demons.

3. A segment which initiates the experimental action (perturbs the world) and creates demons to restore the real world to a quiesent state after a specified time interval.

The process of procedural verification makes extensive use of demons. It is significant that some of these demons are used in somewhat unconventional ways. Typically, demons watch the data base (in this case, the problem solver's world model) and function as consequent and antecedent theorems [9] to maintain and update the data base. As a part of the procedural verification process, a number of these conventional demons operate on the problem solver's world model. Their functions are simple deduction and control structure modification, in a completely standard application. Of interest, however, are the procedural verification or expectation demons which are created to conduct an experiment in the real world and to monitor the system's world model for the expected results of that experiment. They function as follows:

1. Most report, i.e., update the problem solver's world model. Information reported may be data from the real world or conclusions reached because the expectation that created the demon was or was not satisfied in the real world.

2. Some initiate actions or activate other demons after a specified time delay. These may be used to sequence the actions in a single experiment, to terminate actions previously initiated, to watch for a delayed reaction, or to time order a sequence of experiments. A concept may have any number of

independent verification procedures, typically ordered according to their simplicity.

3. All destroy themselves on occurrence of expected events or after a specified period of time.

The lack of real world demons has been a major weakness of execution monitoring systems which are totally dependent upon an accurate world model [4] at all times.

8. DOMAIN DESCRIPTION AND EXAMPLES

The implementation domain for this thesis is the detection and correction of error conditions in the fuel and electrical power systems of a simple twin jet aircraft. It is typical of domains where the problem solver is remote from the actions it directs, having to observe the effects via remote sensors or switches, and where there may be competition (the pilot) for problem solving resources.

The act of problem solving in the cockpit often makes use of default or partial plans in the form of multilevel checklists. A cockpit problem solver may at times have to function despite an inaccurate world model. This possibility arises from sensors that may become inoperative (indicating zero or some fixed quantity despite changes in the real world) or from switches for electrically actuated components that may indicate ON, even though there is no electrical power to the component. The "real world" for this implementation is composed of two separate entities:

1. A real time simulation of the fuel and electrical power distribution systems in a twin engine jet aircraft.

2. The switches and levers in the cockpit that a pilot may turn, push, etc. to control the aircraft.

The "world model" consists of the following:

1. Sensors which indicate fuel quantity, engine temperature, etc.

2. The indications (ON, OFF, IDLE, etc.) provided by the switches, levers, etc. in (2) above.

The world model is that information available to the pilot in the cockpit for making decisions about the performance of the aircraft. Consequently, if the world model is inaccurate, the pilot may observe aircraft system failure symptoms when all of the aircraft systems are operating properly. More disturbingly, he may be denied the ability to observe some failure symptoms even though failure or error conditions exist in the aircraft.

8.1. Example 1

This example will illustrate the system's general strategy for dealing with plan failure, its ability to experiment in the real world, and particularly its ability to distinguish real failure conditions from apparent ones. Initially, the airplane is in level flight and all subsystems are in their normal state, operating properly. The thrust being provided by each engine is 1000 pounds. For some unknown reason, the aircraft's left engine temperature sensor develops an electrical short, causing the temperature indication to be 0 instead of the normal 1700 degrees F., even though the engine continues to operate normally. The electrical short causes the circuit breaker between the DC powered temperature sensor and its DC power source to open. Because the indicated engine temperature is 0, the system's failure detection mechanism believes (incorrectly, it will later be discovered) that the left engine has experienced a flameout. Figure 10 is a chronology of events. It makes a plan, MONITOR1 (Figure 11), to correct the perceived flameout, and executes the plan, resulting in the left restart switch being turned on. Fifty seconds pass with no rise in engine temperature, and the system concludes that its plan for correcting the flameout has failed because SENSOR26 did not respond in a reasonable time.

The system takes a closer look at the requirements for having a normal temperature indication and discovers that the circuit breaker, BREAKER63, to SENSOR26 is open. The system modifies MONITOR1 to produce another plan, MONITOR4, for correcting the flameout, and executes it. MONITOR4, otherwise like MONITOR1, contains a step to close BREAKER63. Fifty more seconds pass without a rise in engine temperature, and the system concludes that MONITOR4 has also failed. Having previously noted that there was a positive fuel flow indication to the troubled engine, the system decides to doubt the indication, and expands the FUEL-FLOW POSITIVE goal to subgoals. Finding that there is an indication that fuel is available and that a fuel path exists from the fuel to the troubled engine, the system is confused momentarily.

```
AT TIME = 0 TURNING (LEFT RESTART) ON

AT TIME = 50

((MONITOR1 FAIL G0002))

ENTERING DIAGNOSIS MONITOR1

FAILPOINT (SENSOR-CHECK ((SENSOR26 TEMPERATURE))
                        TEMP-NORMAL)

(FLAME-OUT ENGINE1) MONITOR1

EXPANDING (TEMPERATURE TEMP-NORMAL) TO (FUEL-FLOW POSITIVE)

ACTION ((TURN BREAKER63 ON)) MONITOR1 50

EXECUTING MONITOR4 INSTEAD-OF ((FLAME-OUT ENGINE1) MONITOR1)


AT TIME = 50 TURNING (LEFT RESTART) ON

AT TIME = 50 TURNING (LEFT TEMPERATURE SENSOR BREAKER) ON

AT TIME = 100

((MONITOR4 FAIL G0005))

ENTERING DIAGNOSIS MONITOR4

FAILPOINT (SENSOR-CHECK ((SENSOR26 TEMPERATURE))
                        TEMP-NORMAL)

(FLAME-OUT ENGINE1) MONITOR4

CONTINUING WORLD3 (TYPE 1)

EXPANDING (FUEL-FLOW POSITIVE) TO
          (AND (FUEL-QUANTITY AVAILABLE)
               (FUEL-PATH EXISTS))
```

Figure 10   Apparent Flameout Example

```
TRYING TO PROCEDURALLY VERIFY SENSOR28 FUEL-FLOW
SUSPENDING FOR AN EXPERIMENT MONITOR4
ADDING EXPECTATION DEMON SENS-CK9 TO
       SENSOR28 FUEL-FLOW ENGINE
ADDING EXPECTATION DEMON SENS-CK10 TO
       SENSOR24 FUEL-QUANTITY TANK
ADDING CONFIRMATION DEMON CONFIRMER11 TO
       SENSOR24 FUEL-QUANTITY
CHANGING THROTTLE1 TO 1100
ADDING TIME-OUT-DEMON TIME-OUT12 TO THROTTLE1
TIME-OUT12 WILL ACTIVATE IN 5 SECONDS
LEFT ENGINE FUEL-FLOW CHANGED TO 140
VERIFIED SENSOR28 FUEL-FLOW
THIS WAS A FALSE FAILURE
THE FOLLOWING SENSORS ARE BROKEN :
    SENSOR26 MONITOR4
    (SENSOR-CHECK ((SEENSOR26 TEMPERATURE)) TEMP-NORMAL)
NEW-ERROR-DEFINITION
   (FIRE (?X ?VIBR ?FUEL-FLOW THRUST-HALF)) ENGINE1
NEW-ERROR-DEFINITION
   (FLAME-OUT (?X ?VIBR ?FUEL-FLOW THRUST-LOW)) ENGINE1
```

Figure 10 Continued

```
(DEFUN MONITOR1 (NUMBER TIME OLD)
       (PROG (NAME MARKERMARKER TIME-ZERO OLD-TIME
              ZZZ)
             (MONITOR-RESTORE 'MONITOR1)
             (COND ((> TIME OLD-TIME)
                    (SETQ OLD-TIME TIME))
                   (T (RETURN NIL)))
             (SETQ
              ZZZ
              (COND
               ((SENSOR-CHECK ((SENSOR26 TEMPERATURE))
                              TEMP-NORMAL)
                (FPROP MONITOR1
                       ((TEMPERATURE TEMP-NORMAL)
                        SENSOR26
                        ENGINE1)
                       EXCEPTIONS)
                (ERROR-REPORT 40.)
                (SETQ MONITORMESSAGE
                      (CONS '(MONITOR1 SUCCESS)
                            MONITORMESSAGE)))
               ((EQ MARKERMARKER 'G0002)
                (COND
                 ((> TIME (/+ TIME-ZERO STANDARD-DELTA))
                  (SETQ
                   MONITORMESSAGE
                   (CONS (APPEND '(MONITOR1 FAIL)
                                 (LIST MARKERMARKER))
                         MONITORMESSAGE)))
                 (T 'SUCCEEDING)))
               (T T
                  (TURN RESTART75 ON)
                  (SETQ TIME-ZERO TIME
                        MARKERMARKER 'G0002)
                  'SUCCEEDING)))
             (MONITOR-SAVE '(NAME MARKERMARKER
                                  TIME-ZERO
                                  OLD-TIME
                                  ZZZ)
                           'MONITOR1)
             (RETURN ZZZ)))
```

Figure 11   Example Plan

All conditions for normal temperature indication are apparently satisfied, but the TEMPERATURE TEMP-NORMAL goal is not satisfied. So, the system decides to conduct an experiment to disambiguate the situation. It adds an expectation demon (SENS-CK9, Figure 12) to watch for changes in fuel flow to the left engine, adds another expectation demon (SENS-CK10) to watch for changes in fuel quantity in the tank feeding ENGINE1, advances the left engine throttle by 10%, and creates a demon to return the throttle to its normal value after 5 seconds.

The purpose of the SENS-CK demons is to report whether the sensors being watched respond as expected. The system also adds confirmation (Figure 13) demons to both fuel quantity sensors. These confirmation demons embody the the dynamic relationship that exists between a change in fuel quantity and fuel flow, i.e., a change in fuel quantity implies fuel flow. The purpose of the confirmation demons is to assert that fuel is flowing if the fuel flow sensor does not respond but the fuel quantity changes, i.e., to discover a broken fuel flow sensor if it exists.

When the system then initiates the dynamic experiment by changing the left engine throttle from 1000 pounds demand to 1100 pounds demand (if the engine is operating properly and all indicators are operating properly), there should be a distinct increase in the fuel flow rate to the left

```
(DEFUN SENS-CK9 (NUMBER NEW OLD)
       (PROG (NP *M TYPE)
             (AND (< NEW (/+ 0 100)) (RETURN T))
             (AND (HERE (LIST 'VERIFIED
                              'SENSOR28
                              '*M)
                        WORLD)
                  (REMOVE-DEMON 'SENS-CK9 3)
                  (DEL-SENS-CK 'SENS-CK9 6 'MONITOR4)
                  (RETURN (CLEAN-UP 'SENS-CK9)))
             (AND (OR (NOT (LOOK-UP 'MONITOR4 DIRECTORY))
                      (> NEW (/+ 10 0 100)))
                  (REMOVE-DEMON 'SENS-CK9 3)
                  (OR (AND (DEL-SENS-CK 'SENS-CK9
                                        6
                                        'MONITOR4)
                           (RETURN T))
                      (COND ((HERE (LIST 'NOT-VERIFIED
                                         'SENSOR28
                                         '*M)
                                   WORLD))
                            (T (STICK-IN (LIST 'NOT-VERIFIED
                                               'SENSOR28
                                               'WORLD6
                                               'MONITOR4
                                               TIME)
                                         WORLD))))
                  (RETURN (CLEAN-UP 'SENS-CK9)))
             (AND
              (OR (AND (SETQ NP (GET 'TANK3 'URR))
                       (EQ 'GREATERP NP))
                  (GREATERP (OLD-VALUE 6) 138))
              (CATCH (STICK-IN (LIST 'VERIFIED
                                     'SENSOR28
                                     'WORLD6
                                     'MONITOR4
                                     TIME)
                               WORLD)
                     VER)
              (UPROP 'MONITOR4
                     (LIST (LIST (SETQ TYPE
                                       (CAR (GET 'SENSOR28
'SENSOR-TYPE)))
                                 (GET 'SENSOR28
                                      'QUANTIZE))
                           'SENSOR28
                           (CAR (GET 'SENSOR28
                                     (GET TYPE 'END))))
                     'EXCEPTIONS)
              (REMOVE-DEMON 'SENS-CK9 3)
              (DEL-SENS-CK 'SENS-CK9 6 'MONITOR4)
              (RETURN (CLEAN-UP 'SENS-CK9)))))
```

Figure 12   Expectation Demon

```
(DEFUN CONFIRMER11 (NUMBER NEW OLD)
       (CATCH (PROG (*M *MT)
                    (AND (< NEW (/+ 100 0)) (RETURN T))
                    (AND (OR (> NEW (/+ 8 0 100))
                             (HERE (LIST 'VERIFIED
                                         (NAME-IT 6)
                                         '*MT)
                                   WORLD6))
                         (REMOVE-DEMON 'CONFIRMER11 3)
                         (CLEAN-UP 'CONFIRMER11)
                         (RETURN T))
                    (AND (HERE (LIST 'VERIFIED
                                     (NAME-IT 4)
                                     '*M)
                               WORLD6)
                         (REMOVE-DEMON 'CONFIRMER11 3)
                         (ADD-IN (LIST 'BROKEN
                                       (NAME-IT 6)
                                       'WORLD6
                                       'MONITOR4
                                       TIME)
                                 WORLD)
                         (CLEAN-UP 'CONFIRMER11)
                         (RETURN NIL)))
              VER))
```

Figure 13   Confirmation Demon

engine, and an increase in the fuel flow rate from TANK3. The expectation demons report whether the expected has happened and result in the modification/execution process being continued or terminated. Then TIME-OUT14 (Figure 14) returns the left engine's thrust to normal after 5 seconds.

In this example, the experiment verifies that the fuel flow sensor is reliable (fuel flow changes from 138 to 139 as a result of the thrust increase). Since FUEL-FLOW POSITIVE is the only active goal and the only precondition for TEMPERATURE TEMP-NORMAL, its verification causes the system to conclude correctly that the flameout indication was false. SENSOR26, the left engine temperature sensor is determined to be broken. This conclusion results in a new failure definition pattern for detecting a fire or a flameout on ENGINE1 in the future. The ?X (matches anything) is placed in the position previously occupied by the temperature indication in the pattern. The effect is that future flameouts of ENGINE1 will be detected by low thrust rather than by low temperature and that fires will be detected by thrust being half its demand value rather than by temperature being high.

```
(DEFUN TIME-OUT12 (NUMBER NEW OLD)
       (PROG NIL
             (AND (< NEW (/+ 100 5)) (RETURN T))
             (MAPCAR (FUNCTION UN-PROTECT)
                     '(59))
             (MAPCAR (FUNCTION (LAMBDA (X Y)
                                        (MESSAGE (LIST
'RETURNING

(NAME-IT X)
                                                             'TO
                                                             Y))
                                        (ALTER X Y)))
                     '(59)
                     '(1000))
             (REMOVE-DEMON 'TIME-OUT12 3)
             (CLEAN-UP 'TIME-OUT12)
             (RETURN T)))
```

Figure 14   Time-Out Demon

8.2. Example 2

This example will illustrate the system's global strategy of skipping information that it perceives to be inaccurate or inconsistent. The skipping technique is only invoked as a last resort, i.e., when all local strategy at all levels has been exhausted and when there are no further subgoals to be expanded.

Initially, the airplane is in level flight with all systems operating properly, when both the left engine thrust sensor and the left engine temperature sensor fall from normal to zero indication. In the process of trying to locate a cause for the apparent flameout (Figure 15), the system is unsuccessful in verifying both the fuel flow and the fuel quantity sensors. Further expansion of the failpoint subgoals reveals that both electrical generators were off (SWITCH99 and SWITCH100), but turning these switches on returns the plan to the original failpoint, FUEL-FLOW POSITIVE. No evident progress has been achieved in executing the plan. But because the system has exhausted its known alternatives (see the Cause-Effect Net, Figure 3) and the failpoint remains, it decides to disregard the requirement to observe fuel flow before turning on the engine restart switch. The modified plan is shown in Figure 16.

```
AT TIME = 0 (((FLAME-OUT ENGINE1) MONITOR1))
AT TIME = 50
((MONITOR1 FAIL G0002))
ENTERING DIAGNOSIS MONITOR1
FAILPOINT (SENSOR-CHECK ((SENSOR28 FUEL-FLOW))
POSITIVE)
(FLAME-OUT ENGINE1) MONITOR1
EXPANDING (FUEL-FLOW POSITIVE) TO
          (AND (FUEL-QUANTITY AVAILABLE)
               (FUEL-PATH EXISTS))
EXPANDING (FUEL-FLOW POSITIVE)
TRYING TO PROCEDURALLY VERIFY SENSOR28 FUEL-FLOW
SUSPENDING FOR AN EXPERIMENT MONITOR1
ADDING EXPECTATION DEMON SENS-CK6 TO
       SENSOR28 FUEL-FLOW ENGINE
ADDING EXPECTATION DEMON SENS-CK7 TO
       SENSOR24 FUEL-QUANTITY TANK
ADDING CONFIRMATION DEMON CONFIRMER8 TO
       SENSOR24 FUEL-QUANTITY
CHANGING THROTTLE1 TO 1100
ADDING TIME-OUT-DEMON TIME-OUT9 TO THROTTLE1
TIME-OUT9 WILL ACTIVATE IN 5 SECONDS
NOT-VERIFIED (SENSOR28 WORLD3 MONITOR1 50)
CONTINUING WORLD3 (TYPE 1)
```

Figure 15 Actual Flameout Example

```
EXPANDING FUEL-PATH EXISTS
EXPANDING (VALVE ON) TO (AND (BREAKER ON) (BUS HOT))
EXPANDING (PUMP-FUEL-ELECTRICAL ON) TO
          (AND (BREAKER ON) (BUS HOT))
TRYING TO PROCEDURALLY VERIFY SENSOR24 FUEL-QUANTITY
SUSPENDING FOR AN EXPERIMENT MONITOR1
ADDING EXPECTATION DEMON SENS-CK12 TO
       SENSOR24 FUEL-QUANTITY TANK
ADDING EXPECTATION DEMON SENS-CK16 TO
       SENSOR24 FUEL-QUANTITY TANK
ADDING TIME-OUT-DEMON TIME-OUT18 TO
       SWITCH35 OVERRIDE77
TIME-OUT18 WILL ACTIVATE IN 15 SECONDS
EXPANDING (BUS HOT) TO
          (OR (AND (GENERATOR HOT) (BREAKER ON))
              (AND (BUS-TIE ON) (BUS HOT)))
EXPANDING (BUS HOT) TO
          (OR (AND (GENERATOR HOT) (BREAKER ON))
              (AND (BUS-TIE ON) (BUS HOT)))
EXPANDING (GENERATOR HOT) TO
          (AND (SWITCH HOT) (VIBRATION NORMAL))
```

Figure 15 Continued

```
THE FOLLOWING CHANGES NEED TO BE MADE (QUOTE :)
    (TURN SWITCH99 HOT)
    (PRIMITIVE-ELEMENT (VIBRATION NORMAL))
    TO MAKE (GENERATOR HOT) (GENERATOR16)
DELAYING INVESTIGATION OF
        ((BUS HOT) ((BUS19) NIL)
         (((SENSOR HOT) ((SENSOR21) NIL))))
        PENDING INVESTIGATION OF OTHER PROBLEMS
EXPANDING (BUS HOT) TO
         (OR (AND (GENERATOR HOT) (BREAKER ON))
             (AND (BUS-TIE ON) (BUS HOT)))
NOT-VERIFIED (SENSOR24 WORLD10 MONITOR1 50)
EXECUTING MONITOR33 INSTEAD-OF ((FLAME-OUT ENGINE1)
MONITOR1)
AT TIME = 50 TURNING (LEFT GENERATOR SWITCH) HOT
AT TIME = 100
((MONITOR33 FAIL G0021))
ENTERING DIAGNOSIS MONITOR33
FAILPOINT (SENSOR-CHECK ((SENSOR28 FUEL-FLOW))
POSITIVE)
(FLAME-OUT ENGINE1) MONITOR33
CONTINUING WORLD22 (TYPE 3)
```

Figure 15 Continued

```
UNVERIFIED PRIMITIVE ELEMENT
    ((VIBRATION NORMAL) (NIL (GENERATOR16))
     (SENSOR NORMAL) (NIL (SENSOR29)))))
CONTINUING WORLD23 (TYPE 0)
CONTINUING WORLD24 (TYPE 0)
CONTINUING WORLD25 (TYPE 4)
EXPANDING (BUS HOT) TO
          (OR (AND (GENERATOR HOT) (BREAKER ON))
              (AND (BUS-TIE ON) (BUS HOT)))
EXPANDING (GENERATOR HOT) TO
          (AND (SWITCH HOT) (VIBRATION NORMAL))
ACTION ((TURN SWITCH100 HOT)) MONITOR33 100
UNVERIFIED PRIMITIVE ELEMENT
  ((VIBRATION NORMAL) (NIL (GENERATOR17))
   (((SENSOR NORMAL) (NIL (SENSOR33)))))
  POSSIBLE THAT (SENSOR NORMAL) (NIL (SENSOR33)) IS
BROKEN
DELAYING INVESTIGATION OF
         ((BUS HOT) ((BUS18) NIL)
          (((SENSOR HOT) ((SENSOR20) NIL))
         PENDING RESULT OF OTHER PROBLEMS
CONTINUING WORLD20 (TYPE 0)
CONTINUING WORLD29 (TYPE 0)
```

Figure 15 Continued

```
CONTINUING WORLD10 (TYPE 0)
UNVERIFIED PRIMITIVE ELEMENT
    ((FUEL-QUANTITY AVAILABLE) ((TANK3) NIL)
    ((SENSOR AVAILABLE) ((SENSOR24) NIL))))
    POSSIBLE THAT (SENSOR AVAILABLE) ((SENSOR24) NIL)
IS BROKEN
CONTINUING WORLD19 (TYPE 4)
CONTINUING WORLD28 (TYPE 4)
EXECUTING MONITOR46 INSTEAD-OF ((FLAME-OUT ENGINE1)
MONITOR33)
AT TIME = 100 TURNING (LEFT GENERATOR SWITCH) HOT
AT TIME = 100 TURNING (RIGHT GENERATOR SWITCH) HOT
AT TIME = 150
((MONITOR46 FAIL G0035))
ENTERING DIAGNOSIS MONITOR46
FAILPOINT (SENSOR-CHECK ((SENSOR28 FUEL-FLOW))
POSITIVE)
(FLAME-OUT ENGINE1) MONITOR46
CONTINUING WORLD35 (TYPE 0)
CONTINUING WORLD36 (TYPE 0)
CONTINUING WORLD39 (TYPE 0)
CONTINUING WORLD40 (TYPE 0)
CONTINUING WORLD38 (TYPE 3)
CONTINUING WORLD42 (TYPE 0)
```

Figure 15 Continued

```
CONTINUING WORLD44 (TYPE 4)
EXPANDING (BUS HOT) TO
          (OR (AND (GENERATOR HOT) (BREAKER ON))
              (AND (BUS-TIE ON) (BUS HOT)))
EXPANDING (GENERATOR HOT) TO
          (AND (SWITCH HOT) (VIBRATION NORMAL))
SKIPPING (SENSOR-CHECK ((SENSOR28 FUEL-FLOW)) POSITIVE)
         IN MONITOR46 AT 150.
EXECUTING MONITOR56 INSTEAD-OF ((FLAME-OUT ENGINE1)
MONITOR46)
SKIPPING (SENSOR-CHECK ((SENSOR28 FUEL-FLOW)) POSITIVE)
AT TIME = 150 TURNING (LEFT RESTART) ON
AT TIME = 151 TEMPERATURE CHANGED TO 1700
AT TIME = 152
SKIPPING (SKIP-SENSOR-CHECK ((SENSOR28 FUEL-FLOW))
POSITIVE)
         PRODUCED PROGRESS
NEW-ERROR-DEFINITION
    (FIRE (TEMP-HIGH ?VIBR ?X THRUST-HALF)) ENGINE1
NEW-ERROR-DEFINITION
    (FLAME-OUT (TEMP-LOW ?VIBR ?X THRUST-LOW)) ENGINE1

AT TIME = 152 LEFT TEMPERATURE NORMAL
((MONITOR56 SUCCESS))
```

Figure 15 Continued

```
(DEFUN MONITOR56 (NUMBER TIME OLD)
       (PROG (NAME MARKERMARKER TIME-ZERO OLD-TIME ZZZ)
             (MONITOR-RESTORE 'MONITOR56)
             (COND ((> TIME OLD-TIME) (SETQ OLD-TIME TIME))
                   (T (RETURN NIL)))
             (SETQ
              ZZZ
              (COND
               ((SENSOR-CHECK ((SENSOR26 TEMPERATURE))
TEMP-NORMAL)
                (FPROP MONITOR56
                       ((TEMPERATURE TEMP-NORMAL) SENSOR26
ENGINE1)
                       EXCEPTIONS)
                (MESSAGE
                 '(SKIPPING (SKIP-SENSOR-CHECK ((SENSOR28
FUEL-FLOW))
                                                 POSITIVE)
                            PRODUCED
                            PROGRESS))
                (NEW-ERROR-DEF 'SENSOR28)
                (SETQ BROKEN (CONS 'SENSOR28 BROKEN))
                (*REMPROP MONITOR46 TRY-SKIP)
                (ERROR-REPORT 50)
                (SETQ MONITORMESSAGE (CONS '(MONITOR56
SUCCESS)
                                           MONITORMESSAGE)))
               ((EQ MARKERMARKER 'G0047)
                (COND
                 ((> TIME (/+ TIME-ZERO STANDARD-DELTA))
                  (SETQ MONITORMESSAGE
                        (CONS (APPEND '(MONITOR56 FAIL)
                                      (LIST MARKERMARKER))
                              MONITORMESSAGE)))
                 (T 'SUCCEEDING)))
```

Figure 16   Modified plan

```
                ((SKIP-SENSOR-CHECK ((SENSOR28 FUEL-FLOW))
POSITIVE)
                 (FPROP MONITOR56
                        ((FUEL-FLOW POSITIVE) SENSOR28
ENGINE1)
                        EXCEPTIONS)
                 (MESSAGE
                  '(SKIPPING (SENSOR-CHECK ((SENSOR28
FUEL-FLOW))
                                                POSITIVE)))
                 (TURN RESTART75 ON)
                 (SETQ TIME-ZERO TIME MARKERMARKER 'G0047)
                 'SUCCEEDING)
                ((EQ MARKERMARKER 'G0046)
                 (COND ((> TIME (/+ TIME-ZERO
STANDARD-DELTA))
                        (SETQ MONITORMESSAGE
                              (CONS (APPEND '(MONITOR56 FAIL)
                                            (LIST
MARKERMARKER))
                                    MONITORMESSAGE)))
                       (T 'SUCCEEDING)))
                (T T
                   (TURN SWITCH99 HOT)
                   (TURN SWITCH100 HOT)
                   (SETQ TIME-ZERO TIME MARKERMARKER 'G0046)
                   'SUCCEEDING)))
              (MONITOR-SAVE '(NAME MARKERMARKER TIME-ZERO
OLD-TIME ZZZ)
                            'MONITOR56)
              (RETURN ZZZ)))
```

Figure 16 Continued

Skipping the fuel flow observation requirement results in the restart being turned on at time 151, the left engine temperature and thrust becoming normal at times 152 and 153, and the plan succeeding. Because the plan succeeds, the system concludes that the fuel flow sensor (SENSOR28) is broken. Skipping has produced progress, in this case, by achieving one or more of the father goals (THRUST THRUST-NORMAL or TEMPERATURE TEMP-NORMAL) of FUEL-FLOW POSITIVE. If skipping had not produced progress, the system would simply give up, having exhausted all its alternatives.

9. SUMMARY AND CONCLUSIONS

.. no matter how clever one is, he never uncovers the real problems by gedankenexperiments. Rather, he thinks awhile, builds a model, runs it, watches it fail, thinks some more, revises it, runs it again, ...

[Chuck Rieger 1975]

The use of underspecified partial or default plans in computer problem solving has received little attention or investigation. A principle deterent to the widespread use of partial plans is the paucity of intelligent execution monitoring systems that respond constructively whenever a plan execution is not successful. The technical barriers to developing intelligent execution monitoring systems are principally the following:

1. The absence of techniques for measuring the relative degree of success or failure of a failed plan.

2. The lack of techniques that relate the degree of success or failure of a plan to what to do about the plan failure.

Straightforward approaches to the first problem, whether a trivial syntactic analysis (counting the number of subgoals achieved) or a more complex semantic analysis (weighting the subgoals achieved, i.e., a hierarchical approach) offer little toward solution of the second problem. Further, ignoring the first problem when reasoning about plan failure results in a binary fail-succeed partitioning of the alternative space; if the plan fails, the only alternative is to discard it and demand that the planner find another approach for achieving the given goal. Consequently, no return for the effort expended in planning and the knowledge gained in the attempted execution is realized.

The fundamental problem is representational. It is difficult to reason about things for which there is no model. Each planner usually has its problem solving strategy intimately bound into its control structure. Consequently, there is no explicit representation of the strategy for achieving a particular goal. In most planners, even if it were possible to represent the strategy explicitly, it would be pointless to associate a strategy with each goal encountered; the strategy is typically the same for every goal. Likewise, planners, with few exceptions [4, 14, 18], view the complete plan as the end product, rather than as a [illegible]cle for execution. The typical representation of a plan [illegible]quence of actions is fine for execution, but is often insufficient alone for understanding the plan's purpose or intent.

But the solution to the problem of using default plans is not as simple as developing a representation for strategy and a separate one for a plan. Rather, the ability for an execution monitor to respond constructively to plan failure can be derived only from a unifying representation -- one that includes strategy as a property of the plan being executed.

Some insight into finding this unifying representation is provided by examining the reasons why a plan fails. To prevent plan failure, planners usually produce complete plans, i.e., every relevant detail is examined in the planning process. In this area, researchers have focussed primarily on insuring that all actions are time sequenced properly and, to a very limited degree, on resolving the physical imprecision in mechanical robot systems. Default plans, even if the contained actions are properly sequenced, may fail because the plan is not detailed enough for the current state of the world. Further, in a reasonably dynamic domain where partial plans are most useful, the computer system's world model may not be either complete or accurate at any given time. So a partial plan fails (we assume the given sequence of steps to be logically correct but perhaps logically incomplete) principally because it does not achieve requisite subgoals, because its execution monitor does not have an accurate picture of a dynamic world, or because both of the above are true. The interaction of missing plan steps and an inaccurate world model produces complications greater than either difficulty alone. When a plan fails or appears to fail, no fixed strategy which emphasizes sufficiency of planning instead of completeness can be guaranteed to find the cause of plan failure in one step. Neither can any local strategy which is keyed by and operates on the local manisfestations of plan failure guarantee single step success. Consequently, the representation for a plan must provide, at least, an appropriate temporal relation among multiple execution attempts.

The results of this research show that a composite strategy (a local strategy initially, reinforced ultimately with a global strategy) which uses the failed plan as a skeleton for further planning can cope successfully with many plan execution failures. The local strategy is sufficient eventually to find the cause of all plan failures resulting from too little detail. Further, it is sufficient, by its ability to experiment in the real world, to identify many apparent plan failures resulting from an inaccurate world model. The composition of this class of problems is highly domain dependent. Generally it consists of those dynamic goals with predictable dynamic side effects whose precise semantics change with time or perturbation. A more global strategy (wherein carefully selected information and requirements which appear to be inconsistent may be ignored in plan execution) is needed to cope with the larger class of the problems which neither detail nor procedural verification illuminate.

The local strategy depends upon the execution monitor's being able to discern where the plan failed, i.e., which subgoal in the plan cannot be accomplished in a sequential execution. It attempts, using a combination of domain dependent tricks, procedural verification, or just patience (waiting for more information) to achieve any subgoals needed to guarantee achievement of the failpoint goal. Procedural verification is especially powerful (when it works), resulting in a catalog of undeniably true information which can be used without further verification or expansion. If none of the above alternatives is not applicable nor successful, the local strategy uses cause-effect reasoning to expand the given goal to more detailed subgoals.

The local strategy was chosen even though, if there is more than one cause of plan failure, it may result in multiple plan execution attempts before success. This keeps the plan modification process in consonance with the original reason for using partial plans -- to obviate unnecessary planning and detail. More importantly, a local strategy can be made essentially independent of the form and content of the failed plan, and it requires no auxilliary knowledge of the decisions made by the planner while constructing the default plan or of the planner's world model at the time of plan construction. It requires only the knowledge of where the plan failed and access to the world model at the time of the plan failed.

In summary, the system to modify a failed partial plan for re-execution has a frugal and flexible character. It plans in only as much detail as necessary to eliminate the local cause of plan failure, and it uses a specifically tailored strategy for each goal it encounters. Its preference for single causes of plan failure not only enables it to avoid the issue of completeness of plans, concentrating instead on sufficiency, but also relieves it of having to cope with data base inaccuracies and inconsistencies that are inconsequential, i.e., unrelated to the immediate failpoint. Though plan failures with more than one cause are naturally more difficult to isolate and correct, the system succeeds in many cases. It is especially good at finding those multiple causes which occur (as a result of the original plan having insufficient detail) in parallel subgoals at the same level of detail and those which occur as part of a single subgoal at different levels of detail. It has difficulty with problems that result from simultaneous world model inaccuracies which are close together, i.e., which occur in brother goals or in a given goal and one of its immediate successor subgoals. Inaccuracies so close together (relative to the goal hierarchy) sometimes interfere with the system's ability to judge its progress. In executing the modified plan, the system refuses to ignore more than one goal until some progress (achieving a predecessor goal) is registered.

Its flexibility, which results from its ability to construct an individual, independent search strategy for each goal it encounters, enables it to concentrate on more than one simultaneous goal. Consequently, it is able to plan each goal to the level of detail needed for consistency, independently of the level of detail for all other goals.

An explicit representation of the general strategy skeleton is used to produce a separately instantiated representation of the particular strategy for each goal encountered. This local stragtegy for each goal encountered is implemented as a CONNIVER-type IF-NEEDED method. This strategy is individually constructed for each goal encountered. It depends on the relationship that exists between the goal and its instantiated preconditions and upon auxilliary properties of the goal concept in the problem domain. It consists of ordered alternatives to be used to achieve the goal. The final alternative in each local strategy is to expand the goal to more detailed subgoals. Since this strategy is not built into the structure of the planner, it can be referenced explicitly and examined. Hence, it is useful, along with the succession of failpoints from unsuccessful execution attempts, in distinguishing among partial success, partial failure, and complete failure.

Reasoning about success or failure requires, in addition to an explicit represention of strategy, an explicit representation of a plan and the results of its execution. The features of the representation used include a hierarchy of data bases; the higher level data bases index the lower level ones by name and control access to them. The data base associated directly with each execution attempt itself contains only highly significant items These highlights provide useful information to all future execution attempts; how to modify the current plan, what redundant actions to avoid, and which goals are active. There are more detailed, lower level data bases, accessible only from the highlights data base associated with the execution attempt. These detailed data bases are linked to each world. Since each world is in one-to-one correspondence with an active goal, a typical world data base contains the representation of the local strategy to achieve the associated goal, saved variable values for re-entry to the strategy, and information accumulated so far in trying to achieve the associated goal. These details are usually significant only to the immediately succeeding execution attempt. Consequently, for the sake of efficiency, the data base hierarchy makes their contents invisible until it is explicitly required.

The merging of the strategy representation and the representation of a plan as a continuing process is the key to successfully using partial plans. At any instant, the model of the plan modification process consists of a linked list of execution attempts, with an explicit representation of the relevant local strategies embedded into each execution attempt. The combination of strategies remaining for each goal and the sequence of failpoints are used to determine whether progress toward the original goal has been achieved. The global strategy chosen, at any point, to continue the process to successful termination, is determined by the semantics of the current goal, the success to present in achieving it, and the presence or absence of remaining local strategy for each goal.

The results of this research may be succinctly summarized as follows:

1. Associating an explicit, individually tailored strategy with each goal encountered enables the system to reason about the success or failure of a plan and to distinguish degrees of each.

2. Imparting the problem solver with an understanding of the dynamics of its domain provides a powerful adjunct to passive reasoning about plan failure. Including an experiment extends the perspective of the local search strategy beyond the local horizon.

3. A simple hierarchy of data bases, the upper level data base being the current plan execution attempt, provides a framework for embedding strategy and indexing alternatives.

4. Allowing the system to conditionally ignore goals for which local strategy has been exhausted without success provides a global view of the plan execution process. This frequently breaks a log jam in the modification and execution process, allowing analysis, modification, and further execution to proceed.

The keys to dealing with plan execution failure are an explicit representation of the strategy for each goal and the merging of that strategy representation into the representation of a plan as a continuing process. Hence, plan failures during execution need not always be catastropic. Frequently, despite an inaccurate world model and the lack of information about how the failed plan was originally created, the plan may be modified locally and be executed successfully. Consequently, because it is possible to deal constructively with plan failure, default plans can be used to enhance the speed and flexibility of the problem solving process.

REFERENCES

1. Computer-Aided Decision Making for Flight Operations, Report 3, Report T-35, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, July 1976.

2. Fahlman, S., "A Planning System for Robot Construction Tasks", AI TR-283, Massachusetts Institute of Technology, 1973.

3. Fahlman, S., "A System for Representing and Using Real World Knowledge", AI Memo No. 331, Massachusetts Institute of Technology, May 1975.

4. Fikes, R.E., "Monitored Execution of Robot Plans Produced by STRIPS", Information Processing 71, Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 1971.

5. Goldstein, I. "Understanding Simple Picture Programs", AI TR-294, Massachusetts Institute of Technology, March 1974.

6. Hayes, Philip J., "A Representation for Robot Plans", Fourth International Joint Conference on Artificial Intelligence, Tblisi, Georgia, USSR, September 1975.

7. Hendrix, G. "Modeling Simultaneous Actions and Continuous Processes", Artificial Intelligence, Vol 4, 1973.

8. Manna, Z. and Waldinger, R. "Knowledge and Reasoning in Program Synthesis", SRI AI Center Technical Note 98, November 1974.

9. McDermott, D.V., "Assimilation of New Information by A Natural Language-Understanding System", AI TR-291, Massachusetts Institute of Technology, February 1974.

10. Minsky, M., "A Framework for Representing Knowledge", in Psychology of Computer Vision", P. Winston Ed., McGraw-Hill, 1975.

11. Moon, D., MACLISP Reference Manual, April 1974.

12. Rieger, C., "An Organization of Knowledge for Problem Solving and Language Comprehension", Artificial Intelligence, Vol 7, 1976.

13. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces", Artificial Intelligence, Vol 15, 1974.

14. Sacerdoti, E., "A Structure for Plans and Behavior" SRI AI Center Technical Note 109, August 1975.

15. Sussman, G. "A Computational Model of Skill Acquisition", AI TR-297, Massachusetts Institute of Technology, August 1973.

16. Tate, A., "INTERPLAN: A Plan Generation System That Can Deal with Interactions Between Goals", Machine Intelligence Unit Memo MIP-R-109, University of Edinburgh, December 1974.

17. Waldinger, R., "Achieving Several Goals Simultaneously", SRI AI Center Technical Note 107, July 1975.

18. Weissman, S., "On a Computer System for Planning and Execution in Incompletely Specified Environments", Ph.D. Thesis, Coordinated Science Laboratory, University of Illinois, June 1976.

19. Winograd, T., "Frame Representations and the Declarative Procedural Controversy", in Representation and Understanding, D. Bobrow and A Collins, Eds., Academic Press, 1975.

20. Winston, P., Notes on Computer Intelligence, Forthcoming.

APPENDIX I.

The fundamental data structure used to record the results of search and to support reasoning about the airplane subsystems is a history [1]. The format for a history is

```
((<PATTERN>) ((<ITEMS SATISFYING <PATTERN>>)
             (<ITEMS NOT SATISFYING <PATTERN>))
            (<SUBHISTORIES IN THE SAME FORMAT AS A
              HISTORY>)).
```

For example,

```
((FUEL-FLOW POSITIVE)((ENGINE1 ENGINE2) NIL)
                    (((SENSOR POSITIVE)((SENSOR28 SENSOR32)
                                        NIL))))
```

means that FUEL-FLOW is POSITIVE for both ENGINE1 and ENGINE2, and that this was determined by observing SENSOR28 and SENSOR32.

VITA

Paul Rodger Davis was born in Union Springs, Alabama on October 5, 1940. He attended the University of Alabama where he earned the B.S. degree and was commisioned a 2nd Lieutenant in the United States Air Force (USAF) in 1963.

In 1970, while on active duty with the USAF, he earned the M.S. degree from the University of Illinois. Between 1970 and 1974, he served abroad with the USAF in South Korea, West Germany, and Italy in a variety of tactical and strategic communications positions. He returned to the University of Illinois in 1974 and earned the Ph. D. degree in 1977.

Married and the father of two sons, he is currently a Major in the USAF. He is a member of Tau Beta Pi, Eta Kappa Nu, Pi Mu Epsilon, and Omicron Delta Kappa.